TOWARDS SECURE COMMUNICATION AND AUTHENTICATION: PROVABLE SECURITY ANALYSIS AND NEW CONSTRUCTIONS

A Dissertation
Presented to
The Academic Faculty

By

Shan Chen

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science

Georgia Institute of Technology

May 2020

TOWARDS SECURE COMMUNICATION AND AUTHENTICATION: PROVABLE SECURITY ANALYSIS AND NEW CONSTRUCTIONS

Approved by:

Dr. Alexandra Boldyreva, Advisor
School of Computer Science
Georgia Institute of Technology

Dr. Mustaque Ahamad
School of Computer Science
Georgia Institute of Technology

Dr. Vladimir Kolesnikov
School of Computer Science
Georgia Institute of Technology

Dr. Paul Pearce
School of Computer Science
Georgia Institute of Technology

Dr. Gaven Watson
Advanced Cryptography
Visa Research

Date Approved: January 8, 2020

ACKNOWLEDGEMENTS

I would like to first thank my advisor Alexandra (Sasha) Boldyreva for her generous support, great patience, and outstanding guidance throughout my doctoral study. I am also very grateful for her understanding and support of my needs besides research, which made my Ph.D. life much easier than it could have become.

I was very fortunate to collaborate with many excellent researchers and talented students. My thanks go to David Pointcheval, Cristina Nita-Rotaru, Manuel Barbosa, Bogdan Warinschi, and my mentor Gaven Watson at Visa Research, as well as student co-authors Pierre-Alain Dupont, Samuel Jero, and Matthew Jagielski.

I thank my proposal and thesis committee for their valuable time and helpful advice. I would also like to thank John Steinberger for introducing me to the fascinating world of cryptography and steered me in the right way towards establishing good research abilities.

Finally, I would express my sincere gratitude to my parents and my girlfriend for their constant love and encouragement. I thank my labmates and all my dear friends for their support and wish them all the best in their lives.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Secure communication and authentication are some of the most important and practical topics studied in modern cryptography. Plenty of cryptographic protocols have been proposed to accommodate all sorts of requirements in different settings and some of those have been widely deployed and utilized in our daily lives. For instance, over half of web traffic is now protected by the Transport Layer Security (TLS) protocol to encrypt the communication between web servers and clients. Not surprisingly, these real-world protocols become hot targets of malicious attacks, which could lead to disastrous confidential information leakage and significant financial loss. It is therefore a crucial goal to provide formal security guarantees for such protocols.

In this thesis, we apply the provable security approach, a standard method used in cryptography to formally analyze the security of cryptographic protocols, to three problems related to secure communication and authentication:

First, we focus on the case where a user and a server share a secret and try to authenticate each other and establish a session key for secure communication, for which we propose the first user authentication and key exchange protocols that can tolerate strong corruptions on the client-side.

Next, we consider the setting where a public-key infrastructure (PKI) is available and propose models to thoroughly compare the security and availability properties of the most important low-latency secure channel establishment protocols: TLS 1.3 over TCP Fast Open (TFO), Quick UDP Internet Connections (QUIC) over UDP, and QUIC[TLS] (a new design for QUIC that uses TLS 1.3 key exchange) over UDP.

Finally, we perform the first provable security analysis of the new FIDO2 protocols, the promising proposed standard for passwordless user authentication from the Fast IDentity Online (FIDO) Alliance to replace the world's over-reliance on passwords to authenticate users, and design new constructions to achieve stronger security.

CHAPTER 1
INTRODUCTION

1.1 Motivation and Goal

Secure communication and authentication are some of the most important and useful tools provided by cryptography for information security. In particular, secure communication prevents private communication between two entities from being eavesdropped and modified by a third party while authentication verifies the identity of an entity.

So far, there are plenty of cryptographic protocols proposed by both cryptographers and practitioners to accommodate all sorts of requirements in different settings. Among them some protocols have been widely deployed and utilized in our daily lives. As a prominent example, the Transport Layer Security (TLS) [1] protocol is currently used to encrypt over half of web traffic to protect the communication between web servers and clients, which in particular guarantees the confidentiality and integrity of the browsed web content if the website employs HTTPS [2]. TLS also provides server authentication by default, which for instance prevents network attackers from impersonating a legal website.

Not surprisingly, such real-world protocols are always hot targets of malicious attacks. To mitigate potential attacks and construct secure protocols, it is important to perform *formal analyses* of the cryptographic protocols to evaluate their security guarantees and identify possible vulnerabilities.

Again taking TLS for example, there have been a lot of reported attacks [3] that exploited its design flaws and implementation errors, resulting in disastrous confidential information leakage and significant financial loss. An important cause of those exploited vulnerabilities was the insufficient formal security analysis of the previous TLS standards. Since 1999, several TLS standards have been designed and released by the Internet Engineering Task Force (IETF) TLS working group. However, their security was mostly "believed" rather than guaranteed in a formal way until the first thorough security analysis of TLS 1.2 appeared in 2012 [4]. In order to enhance the security (and performance) of the current deployed standard TLS 1.2 [1], the new standard TLS 1.3 [5] was designed in an unprecedented joint effort with the academic community. With extensive formal analyses performed by cryptography and security researchers along with its design process [6, 7, 8, 9, 10, 11, 12, 13, 14, 15], TLS 1.3 upon its release has very strong security guarantees in contrast with its predecessors.

Our goal in this thesis is to provide formal cryptographic analyses of some of the most important secure communication and authentication protocols and propose new constructions to achieve stronger security in some settings. In particular, we apply the *provable security approach*, a standard methodology in modern cryptography to formally analyze the security of a cryptographic protocol, to three problems related to secure communication and authentication. We hope our results will help the practitioners understand the security guarantees and vulnerabilities of the existing protocols and facilitate the design and deployment of more secure and efficient constructions by proposing candidate protocols with strong provable security. Our contributions are sketched below.

1.2 Contributions

Let us first briefly recall the provable security approach, which is used throughout this thesis. The idea of provable security dates back to the seminal work of Goldwasser and Micali [16] and now this approach is widely applied to analyze all kinds of cryptographic protocols. It consists of three main steps: first formalizing the general protocol syntax that captures the target protocol to be analyzed, then defining the security model to specify the security goals and adversarial abilities, and finally based on the defined model proving the security of the target protocol or identifying attacks that break the security. The proofs proceed by reduction, i.e., showing that if an attacker breaks the security of the target protocol then the underlying computational hardness assumption (or the security) of some building block does not hold.

1.2.1 Human Authenticated Key Exchange

Motivation. Our first problem concerns the case where a user and a server share a secret and try to authenticate each other and establish a session key for secure communication. This problem has been extensively studied under the topic *Password-Authenticated Key Exchange (PAKE)*, since the seminal work by Bellovin and Merritt [17]. However, what happens if the terminal through which the user communicates with the server has been compromised? The terminal may have a spyware keylogger recording the user's keystrokes and sending them to the attacker.

The existing security definitions for PAKE acknowledge the problem by modeling *strong corruptions* when the adversary corrupts the user's terminal and learns all of its current state. However, none of the existing protocols try to offer any security for the user's future sessions in this case. Basically, the consensus is that in case of strong corruption, all is lost to the user, and the only thing guaranteed is that this should not violate the security of his past sessions and other users. Indeed, cryptography cannot do much since the attacker invading a machine would know everything, as it can read all secrets being stored or typed.

In this work, we take a fresh look at this problem and try to provide a solution. That is, if a user's terminal has been fully compromised, can we still protect his (past and future) sessions from other trusted terminals, even though the *same* long-term secret is used?

Basic Approach. Our basic idea is to store *no* long-term secrets on the terminals, and instead, employ human computation or an additional device such as RSA SecurID to boost security. (We think it is much more reasonable to assume that the human and the offline device stay uncompromised than the terminals and other devices used for connecting to servers.) In a bit more detail, we ask the server to send a random challenge, then let the human user generate a response by computing (in his head or with an additional device) a function of the memorized long-term secret and enter it into the terminal. Then we can use a PAKE-like protocol ran on the response as a common ephemeral secret, also known as a *one-time password*.

A PAKE protocol, that is usually used to prevent off-line dictionary attacks, here provides the guarantee that no information is leaked about the one-time passwords in passive and even active sessions. It is important to limit the information leakage about the long-term secret of the user, since one-time passwords, were they in the clear, could have helped recovering the long-term secret. This is unfortunately the case when they are generated with functions that are easy enough to be computed by a human. On the other hand, if an additional device is used to derive the one-time passwords, their privacy may be less critical and so resistance to off-line dictionary attacks is not required anymore, which allows the use of a weaker variant of PAKE.

Contribution Overview. We propose the first user authentication and key exchange protocols that can tolerate *strong corruptions* on the client-side. More specifically, our protocols guarantee the following: even if a user happens to log in to a server via a *fully compromised* terminal, his other past and future sessions executed through honest terminals still remain secure. Note that such guarantee is impossible if the user simply types his secret into the terminal.

We first define the security model for *Human Authenticated Key Exchange (HAKE)* protocols and then propose two generic HAKE protocols based on the *Human-Compatible (HC)* function family, PAKE, commitment scheme, and authenticated encryption. We prove that our HAKE protocols are secure against strong corruptions under reasonable assumptions.

All of the building blocks like PAKE have existing efficient and secure instantiations, except for the new primitive HC function families that we propose. Security-wise, the adversary should be able to see multiple challenge-response pairs, among which some of the challenges could be chosen by the attacker (adaptive queries vs. non-adaptive queries).

This is because the attacker, who compromises a terminal, can eavesdrop on the communication with the human user. And an active attacker who took control of the terminal can impersonate the server and ask the user to answer maliciously chosen challenges. But still, the adversary should not be able to forge a valid response for a new random challenge, so that future sessions remain safe.

Finding such a function family would be easy if we did not have the human-computability restrictions. We survey some works on secure human-based computation later, but they are not directly suitable for us. Luckily, a recent paper by Blocki, Blum, Datta, and Vempala [18] (almost) provided a solution. They proposed a way for a human user to authenticate to a computer that does not offer privacy (honest-but-curious). Such a computer stores a set of challenges and the user authenticates by providing a response to a random challenge. In their concrete construction, a challenge is a set of images, the secret user memorizes is a correspondence between images and numbers and the response is some basic function using addition of the digits (modulo 10). The authors provide experimental evidence that their scheme can be used by a human user. Namely, the secret can be memorized and the response can be computed within reasonable time by an average human user. The authors also propose a tool to help secret memorization. While the usability of their solution is not perfect, it is definitely a start and further research will hopefully yield protocols with better usability.

Although the construction and security results from [18] are very useful for our work, we cannot use them as is. The problem is that it is not known whether security of their scheme holds when the attacker can see responses to maliciously chosen challenges (adaptive queries). We extend their analysis and prove a second conjecture that the unforgeability of their HC function family still holds if the adversary can make very few adaptive queries.

Our Confirmed HAKE (one of the two generic HAKE protocols) is designed to rely on those HC function families (whose security can tolerate some adaptive queries): after the PAKE completion using the first response, the human user selects a random challenge and enters it into the terminal, who encrypts the challenge under the recently established session key and forwards the result to the server. The server decrypts, computes the response, and sends it, also encrypted to the terminal. The terminal decrypts and displays the response and the human user verifies it. If verification fails, the user needs to take measures against suspected terminal infection and possibly abort the long-term secret. Encrypting the terminal-server communication here is needed for authenticity in case of an honest terminal, to prevent a network adversary to ask the server maliciously chosen challenges. We show that this extended protocol limits the number of responses the attacker infecting the terminal can obtain for malicious challenges of its choice (in that case, the adversary will not be able to make the user pass the connection confirmation step). We argue that this addition, while adds a little bit more work for the human user, does not violate human computability for our instantiation, i.e., that the user can select a random challenge and verify the response. Furthermore, we show that the Confirmed HAKE provides explicit authentication assuming that the encryption scheme is secure authenticated encryption.

Finally, we consider the more practical case where the human user is allowed to get help from an offline device such as RSA SecurID. Such device-assisted HAKE protocols are very efficient as the human user does not need to do any computations. Furthermore, an additional device permits simple HC function family instantiations with stronger security (e.g., pseudorandom functions): leaking information about several challenge-response pairs might not endanger the long-term secret. As a result, we can rely on a weaker variant of PAKE and get device-assisted HAKE protocols that are even more efficient.

Open Problems. We hope that our work will stimulate further results about secure human-compatible cryptographic function families. We leave to future works to formally prove the unforgeability property (against several adaptive queries) of the HC function from [18], and possibly finding other HC function families with such security. Those would allow to avoid additional devices and still have a completely proven efficient HAKE protocol. Improving the usability of the scheme from [18] will indeed imply improved HAKE protocols, and may have other applications. Another interesting question is to design a coin-flipping protocol with a human participant. Such protocol could be used within HAKE to prevent the attacker to ask malicious challenges. Eventually, after this first step of modeling HAKE protocols with symmetric long-term secrets shared between the user and the server, asymmetric secrets would be important to consider. This would be similar to the so-called *verifier-based PAKE* that helps moderate the impact of corruption of the server.

1.2.2 Comparing TLS 1.3 (over TCP Fast Open) to QUIC

Motivation. We next switch our focus on the setting where a *public-key infrastructure (PKI)* is available. As mentioned before, secure channel establishment protocols such as TLS are some of the most important cryptographic protocols, enabling the encryption of Internet traffic. Currently, more than half of all Internet traffic is encrypted according to a 2017 EFF report [19], with Google reporting that 94% of its traffic is encrypted as of October 2019 [20]. This trend has also been facilitated by efforts like the free digital certificate issuer Let's Encrypt servicing 87 million active (unexpired) certificates and 150 million unique domains at the end of 2018 [21].

Secure channel establishment protocols allow two parties (where one or both parties have a public key certificate) to establish a secure communication channel over the insecure Internet. Typically, the parties first authenticate all parties holding a public key certificate and agree on a session key — the key exchange phase. Then, this session key is used to encrypt the communication during the session — the secure channel phase.

The main secure channel establishment protocol in use today is TLS. The session key establishment with TLS today involves 3 round-trip times (RTTs) of end-to-end communication, including the cost of establishing a TCP connection before the TLS connection. Further, this TCP cost is paid every time the two parties communicate with each other, even if the connection is interrupted and then immediately resumed. Given that most encrypted traffic is web traffic, this cost represents a significant performance bottleneck, a nuisance to users, and financial loss to companies. For instance, back in 2006 Amazon found that every 100ms of latency cost them 1% in sales [22], while a typical RTT on a connection from New York to London is 70ms [23].

Not surprisingly, many efforts in recent years have focused on reducing latency in secure channel establishment protocols. The focus has been on reducing the number of interactions (or RTTs) during session establishment and resumption without sacrificing much security. The most important protocols addressing this goal are TLS 1.3 [5] (the just-released successor to the current TLS 1.2 standard) and Google's QUIC [24].

With TLS 1.3, it is possible to reduce the number of RTTs (prior to sending encrypted data) during session resumption to 1. This reduction is achieved by utilizing a session ticket that was saved during a previous communication and multiple keys (which we call stage keys) that can be set within one session, of which some keys are set faster (with slightly less security) so that data can be encrypted earlier. The remaining 1-RTT during session resumption is due to the aforementioned TCP connection. However, one recent optimization for TCP, called TCP Fast Open (TFO) [25, 26] extends TCP to allow for 0-RTT resumption connections, so that the client may begin data transmission immediately. The mechanism underlying this optimization is a cookie saved from previous communication, similar to the ticket used by TLS 1.3.

Like TLS 1.3, Google's QUIC uses weaker initial keys, under which data can be encrypted earlier, and a token saved from previous communication between the parties. But unlike TLS, QUIC operates over UDP rather than TCP. Instead of relying on TCP for reliability, flow control, and congestion control, QUIC implements its own data transmission functionality, integrating connection establishment with key exchange. These features allow QUIC to have 1-RTT full connections and 0-RTT resumption connections.

In Table 1.1 we show the cost of establishing full and resumption connections for several layered protocol options achieving end-to-end security. These include TLS 1.2 over TCP, TLS 1.3 over TCP, TLS 1.3 over TFO, QUIC over UDP, and the new design for

Table 1.1: Latency comparison of layered protocols

Layered Protocol	Full Connection	Resumption Connection
TCP+TLS 1.2	3-RTT	2-RTT
TCP+TLS 1.3	2-RTT	1-RTT
TFO+TLS 1.3	2-RTT	**0-RTT**
UDP+QUIC	1-RTT	**0-RTT**
UDP+QUIC[TLS]	1-RTT	**0-RTT**

QUIC [27] (which we refer to as QUIC[TLS] [28] to indicate that it borrows the key exchange from TLS 1.3) over UDP. It is clear that the last three win in terms of the number of interactions. But how does their security compare?

At first glance, the question is easy to answer. Recent works have done formal security analyses of TLS 1.3 [29, 30, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] and Google's QUIC [31, 32]. Most works confirm that (the cryptographic cores of) both protocols are provably secure under reasonable computational assumptions. Moreover, as shown in [32, 11], their 0-RTT data transmission designs cannot achieve the same strong security guaranteed by classical key exchange protocols with at least one RTT. In particular, the 0-RTT keys do not provide forward secrecy and the 0-RTT data suffers from replay attacks. Overall, it might seem that all three layered protocols mentioned above are equally secure.

However, a closer look reveals that the answer is not that simple. First, all aforementioned formal security analyses, except for [32] analyzing the IP spoofing (source validation) of QUIC, did not consider packet-level availability attacks. Therefore, it is not clear at the packet level what security can be achieved and what attacks can be prevented by these protocols. In other words, we have no formal understanding of what security can be obtained when layering protocols. Also, TFO uses some cryptographic primitives, such as a cookie, to prevent IP spoofing, but, to the best of our knowledge, no formal analysis has been done. Furthermore, the security of QUIC[TLS] has not been formally analyzed (although some security aspects can be reduced to those of Google's QUIC and TLS 1.3).

Contribution Overview. To compare their security, we develop novel security models that permit "layered" security analysis. In addition to the standard goals of server authentication and data privacy and integrity, we consider the packet-level availability attacks that are not usually taken into account by existing security models that focus mainly on the cryptographic cores of the protocols. Our models are sketched as follows.

Like most security models, we consider a very powerful attacker who can initiate communications between honest parties, can intercept, inject, drop, or modify the exchanged packets, and can adaptively learn parties' stage keys or adaptively corrupt them to learn their long-term keys and secret states. The attacker can also have prior knowledge of the exchanged data. However, the attacker should not be able to prevent clients from establishing final session keys without noticing the attacker's involvement (Server Authentication) or using these keys to achieve a secure channel with data privacy and integrity (Channel Security). These standard security goals are captured by our first model.

For the second model that deals with packet-level availability attacks, we first follow QACCE [32] to consider IP-spoofing prevention (also known as address validation) and further extend it to additionally capture IP-spoofing attacks in the full connections. Then, we design several novel notions for packet-level authentication as follows.

First, we define Header Integrity to capture the integrity of the whole unencrypted packet header. (Note that previous models like QACCE only cover the header integrity implied by the authenticity security of the underlying authenticated encryption scheme.) To enable fine-grained security analyses and comparisons, we split the above notion into two related ones, Key Exchange (KE) Header Integrity and Secure Channel (SC) Header Integrity, which capture header integrity during the key exchange phase and secure channel phase respectively. Furthermore, we define the notion of KE Payload Integrity to cover availability attacks that modify the payloads of packets sent during key exchange. We note that unlike the availability attacks shown in [32], successful attacks under our new notions do not affect the client's session key establishment and therefore are harder or impossible

Table 1.2: Security comparison

	TLS 1.3 +TFO	QUIC +UDP	QUIC[TLS] +UDP
0-RTT Key Forward Secrecy [11]	✗	✗	✗
0-RTT Data Anti-Replay [11]	✗	✗	✗
Server Authentication	✓	✓	✓
Channel Security	✓	✓	✓
IP-Spoofing Prevention	✓	✓	✓
KE Header Integrity	✗	✗	✗
KE Payload Integrity	✓	✗	✗
SC Header Integrity	✗	✓	✓
Reset Authentication	✗	✗	✓

to detect by the client. This makes such attacks more harmful and their treatment more important. Finally, we formalize the new goal of Reset Authentication to deal with attacks forging a reset packet to abruptly terminate an honest party's session.

Equipped with our new models we provide a detailed comparison of TLS 1.3 over TFO, QUIC over UDP, and QUIC[TLS] over UDP. In particular, we show that TFO's cookie mechanism does provably achieve the security goal of IP spoofing prevention. Additionally, we find several new availability attacks that manipulate the early key exchange packets without being detected by the communicating parties.

Our results are summarized in Table 1.2. As mentioned above, by [11] results, no protocol achieves forward secrecy for 0-RTT keys or protects against 0-RTT data replays (which contribute to the first two rows in the table). By including packet-level attacks in our analysis, our results shed light on how the reliability, flow control, and congestion control of the above layered protocols compare, in adversarial settings.

Even though QUIC may not be able to sustain the competition in the long run despite stronger security, we hope our models will help practitioners better understand the advantages and limitations of novel secure channel establishment protocols.

1.2.3 Provable Security Analysis of FIDO2

Motivation. Our final work is related to the ongoing effort by the Fast IDentity Online (FIDO) Alliance to replace the world's over-reliance on passwords to authenticate users.

Authentication is used to verify the identity of an entity and plays an important role in ensuring authorized access to confidential data or services. Currently, with the help of a PKI a server can be easily authenticated by having its associated certificate validated, whereas a user typically authenticates himself using a *low-entropy* human-memorizable password.

With massive password leakage reported by recurring data breaches [33], it becomes well-acknowledged that strong security cannot be achieved with passwords only. According to Verizon's 2019 Data Breach Investigations Report [34], 81% of data breaches are caused by compromised, weak, and reused passwords. To deal with the problems caused by dominant password usage, various authentication solutions (see [35] for a 2012 survey) have been proposed towards replacing passwords, but their wide adoption still falls behind. As pointed out in [35], "many academic proposals failed to gain traction because researchers rarely consider a sufficiently wide range of real-world constraints". In particular, some solutions that provide strong security are expensive to deploy or difficult to use.

In 2013, an open industry association called the Fast IDentity Online (FIDO) Alliance was launched and its mission is to provide authentication standards to help reduce the world's over-reliance on passwords to authenticate users. Unlike most academic proposals that mainly focus on strong security, FIDO also aims at fulfilling the practical requirements for smooth deployment and satisfying usability. Currently, FIDO is driven by hundreds of global technology leaders including Google, Amazon, Alibaba, Intel, Microsoft, Paypal, Samsung, Visa, and major banks. It is truly a world-wide effort with growing impact: FIDO has member-driven working groups in the US, China, Europe, India, Japan, and Korea.

To address various authentication use cases, the FIDO Alliance has published three sets of specifications [36]: FIDO Universal Second Factor (FIDO U2F), FIDO Universal Authentication Framework (FIDO UAF), and FIDO2. FIDO U2F supports second-factor (or multi-factor) authentication that augments the security of the existing password infrastructure by adding a strong authentication factor, while FIDO UAF is designed for passwordless authentication. FIDO2 is a newly proposed standard that unifies and improves the previous specifications to maximize platform support. It consists of the World Wide Web Consortium's (W3C) Web Authentication (WebAuthn) specification and FIDO Alliance's Client-to-Authenticator Protocol (CTAP), where CTAP consists of CTAP1 as a new name for FIDO U2F and the main component CTAP2.

FIDO protocols are moving towards wide deployment and standardization with great success. In January 2017, Facebook announced support for FIDO Authentication, which brought the number of potential FIDO users to more than 3 billion. From December 2017, FIDO Authentication is backed by hardware key attestation openly available on any Android 8.0 or later device. By April 2018, major web browsers (e.g., Chrome, Firefox, Edge, etc.) have implemented the FIDO standards. In December 2018, FIDO UAF 1.1 and CTAP were recognized as international standards by the International Telecommunication Union's Telecommunication Standardization Sector (ITU-T). In 2019, WebAuthn became an official web standard and Android and Windows Hello earned FIDO2 Certification.

As a promising future standard for user authentication, FIDO protocols are expected to achieve strong security. Descriptions of their security goals, possible attacks, and countermeasures are provided in the published specifications [36]. However, these descriptions are of course informal and can only serve as security guidelines. To understand the exact security guarantees and fix any potential vulnerabilities before wide deployment of the FIDO protocols, it is critical and urgent to provide their formal security analyses.

Unfortunately, as we will survey later in Chapter 5, there are not much security analysis about FIDO protocols. In particular, there is *no* provable security analysis of the new

FIDO2 protocols despite their fast deployment process. Our goal is to provide the first provable security analysis of the latest FIDO2 protocols to help practitioners understand their security guarantees and vulnerabilities. More precisely, we focus on the main FIDO2 components WebAuthn [37] and CTAP2 [38].

FIDO2 Overview. Before describing our contributions, we first recall the high-level ideas behind the FIDO2 protocols.

WebAuthn is a web API that can be built into browsers and related web platform infrastructure to enable web applications (of online services) to integrate public-key user authentication. The core component of WebAuthn is a *passwordless* "challenge-response" scheme between a secure device owned by the user (e.g., a security token or a smartphone) and a server, which is used for both user registration and authentication. Such a device-assisted "challenge-response" scheme works as follows: first the server sends a random challenge to the secure device through a client (e.g., a browser or an operating system installed on the user's laptop), then the device replies with an unforgeable signature associated with the challenge (and a public key used for future authentication if in the registration phase), and finally this signature is verified by the server. The security of the above scheme may seem straightforward at first glance, but subtleties are incurred by the involvement of multiple parties (i.e., users, devices, clients, and servers). To achieve the desired overall security, one needs to carefully design the functions and identify the security requirements for each party.

On the other hand, CTAP2 specifies the communication between a remote authenticator (i.e., the secure device) and a client that has a user PIN as input. Roughly speaking, the security goal of CTAP2 is to set up the authenticator with the user's PIN and "bind" a trusted client to the set-up authenticator such that the authenticator accepts only messages sent from a "bound" client.

Contribution Overview. To better understand and improve each component protocol, our analysis is conducted in a *modular* way, i.e., we analyze CTAP2 and WebAuthn separately and then derive the overall security of FIDO2 by composing their security.

First, we design a model to analyze the more complex CTAP2 protocol. We show that its simpler but as secure version is provably secure, in the sense that an authenticator can only be accessed through the client-to-authenticator authenticated channels established from the CTAP2 "binding" process. However, if any of the channels towards the same authenticator is compromised, the attacker can have access to other channels due to the *same* authenticator secret used to "bind" multiple clients. Besides, CTAP2's "binding" process is not resistant to man-in-the-middle attacks. To address these issues, we propose an efficient protocol that employs password-authenticated key exchange for the "binding" process and prove its security in the strong sense.

Next, we analyze the user authentication security of the WebAuthn protocol based on our second model. We propose two security notions and show that WebAuthn can only be proved secure for the weak security, which guarantees that any successful user authentication must involve the registered authenticator and go through a trusted client (i.e., a client that has access to an authenticated channel towards that authenticator). However, WebAuthn is not secure if any of the trusted clients is compromised, even when the user tries to authenticate to the server via a secure trusted client (i.e., not affected by the compromised client). This is because the user cannot detect if his authenticator accepts messages through an unexpected channel. We hence augment WebAuthn to add such user detectability and prove its strong security.

We hope our models and provable security results will help clarify the security guarantees of the FIDO2 protocols and expect our proposed constructions to facilitate the design and deployment of more secure passwordless user authentication protocols.

1.3 Declaration of Co-Authorship and Previous Publications

Our investigation of human authenticated key exchange was published in CSF 2017 with co-authors Alexandra Boldyreva, Pierre-Alain Dupont, and David Pointcheval [39]. This work was also included in Pierre-Alain Dupont's dissertation [40]. He mainly contributed to the device-assisted HAKE protocols and the security proofs, while I mainly contributed to the confirmed HAKE protocol and the HC function family.

Our security results of comparing low-latency secure channel establishment protocols were published in ESORICS 2019 with co-authors Samuel Jero, Matthew Jagielski, Alexandra Boldyreva, and Cristina Nita-Rotaru [41].

Our security analysis of FIDO2 is currently in submission, which is collaborated with Manuel Barbosa, Alexandra Boldyreva, and Bogdan Warinschi.

1.4 Road Map

We first specify the notations and recall the cryptographic building blocks used in this thesis in Chapter 2. Our detailed investigations of the three problems sketched above are respectively presented in Chapter 3 4 5. Finally, we conclude in Chapter 6.

CHAPTER 2
PRELIMINARIES

2.1 Notations

Let $\{0,1\}^*$ denote the set of all finite-length binary strings (including the empty string ε) and $\{0,1\}^n$ denote the set of n-bit binary strings. $[n]$ denotes the set of integers $\{1,2,\dots,n\}$. For a finite set $\mathcal{R}$, let $|\mathcal{R}|$ denote its size and $r \overset{\$}{\leftarrow} \mathcal{R}$ denote sampling r uniformly at random from $\mathcal{R}$. For a binary string s, let $|s|$ denote its length in bits. $y \leftarrow F(x)$ (resp. $y \overset{\$}{\leftarrow} F(x)$) denotes y being the output of the deterministic (resp. probabilistic) function F with input x. Let $x \leftarrow a$ denote assigning value $\mathcal{A}$ to variable x. We use the wildcard $\cdot$ to indicate any valid input of a function.

Let λ denote the security parameter. We say an algorithm or a function is *efficient* if its running time is polynomial in the security parameter λ. We say a function or a probability is *negligible* (denoted by $\mathsf{negl}(\lambda)$) if it decreases faster than the inverse of any positive polynomial of λ. We say a security notion is achieved if its associated advantage $\mathbf{Adv}(\mathcal{A})$ is small (e.g., negligible) for any *probabilistic polynomial time (PPT)* adversary $\mathcal{A}$ with reasonable resources (e.g., with certain number of queries to the given oracle).

2.2 Pseudorandom Function

For a function family $F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^m$, consider the following security experiment associated with an adversary $\mathcal{A}$. In the beginning, sample a bit $b \overset{\$}{\leftarrow} \{0,1\}$. If $b = 0$, $\mathcal{A}$ is given oracle access, i.e., can make queries, to $F_k(\cdot) = F(k,\cdot)$ where $k \overset{\$}{\leftarrow} \{0,1\}^\lambda$. If $b = 1$, $\mathcal{A}$ is given oracle access to $f(\cdot)$ that maps elements from $\{0,1\}^n$ to $\{0,1\}^m$ uniformly at random. In the end, $\mathcal{A}$ outputs a bit b' as a guess of b. The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_F^{\mathsf{prf}}(\mathcal{A}) = |\Pr[b' = 1|b = 0] - \Pr[b' = 1|b = 1]|$, which measures

$\mathcal{A}$'s ability to distinguish F_k (with random k) from a random function f.

F is a *pseudorandom function (PRF)* if: 1) for any $k \in \{0,1\}^\lambda$ and any $x \in \{0,1\}^n$, there exists an efficient algorithm to compute $F(k,x)$; and 2) for any PPT adversary $\mathcal{A}$, $\mathbf{Adv}_F^{\mathsf{prf}}(\mathcal{A}) = \mathsf{negl}(\lambda)$.

A *weak* PRF is weaker than a PRF in the sense that the adversary $\mathcal{A}$ cannot get $F_k(x)$ for arbitrary xs but only have access to pairs $(x, F_k(x))$ for random xs.

2.3 Commitment Scheme

Roughly, a commitment scheme allows a user to "commit" a value such that the receiver cannot learn any information about it, but the user is also not able to change his mind later.

Syntax. A (non-interactive) commitment scheme $\mathcal{CS}$ is a triple $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open})$ such that Setup generates the global public parameters and the other two algorithms are defined as follows:

- $\mathsf{Com}(x)$: on input a message x, and some internal random coins, it outputs a commitment c together with an opening value s;
- $\mathsf{Open}(c,s)$: on input a commitment c and then opening value s, it outputs either the committed value x or $\perp$ in case of invalid opening value.

Correctness requires that for every x, if $(c,s) \leftarrow \mathsf{Com}(x)$, then $\mathsf{Open}(c,s)$ outputs x.

Security. The usual security notions for commitment schemes are the *hiding* property, which says that x is hidden from c, and the *binding* property, which says that once c has been sent, no adversary can open it in more than one way. We will also need additional properties, such as *extractability* (a simulator can extract the value x to which c will be later opened) and *equivocality* (a simulator can generate some fake commitments c it can later open to any x). These features are provided from *trapdoors*, generated by an alternative setup algorithm and privately given to the simulator.

In this thesis, we will simply let $\mathbf{Adv}_{\mathcal{CS}}(\mathcal{A})$ denote the advantage an adversary can get against any of these security notions. We refer to [39] for detailed security definitions.

Instantiation. An efficient instantiation, with all our expected security properties, in the *random oracle model* [42], can be described as follows:

Given a hash function $\mathcal{H}$ onto $\{0,1\}^\lambda$ modeled as a random oracle, we have:

- $\mathsf{Com}(x)$: Generate $r \overset{\$}{\leftarrow} \{0,1\}^{2\lambda}$ and output $(c \leftarrow \mathcal{H}(x,r), s \leftarrow (x,r))$;
- $\mathsf{Open}(c, s = (x,r))$: if $\mathcal{H}(s) = c$, return x, otherwise, return $\perp$.

In the random oracle model, this simple scheme is trivially computationally *binding* (because $\mathcal{H}$ is collision-resistant) and statistically *hiding* (because for a large domain $\{0,1\}^{2\lambda}$ for r there are almost the same number of possible rs for any x that would lead to the same commitment c). Additionally, we can use the programmability of the random oracle for equivocality and the list of query-answers for extractability. More details can be found in [39].

2.4 Message Authentication Code

For a (deterministic) *message authentication code (MAC)* $\mathsf{MAC} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n$, consider the following security experiment associated with an adversary $\mathcal{A}$. In the beginning, sample a random key $k \overset{\$}{\leftarrow} \mathcal{K}$. Then, $\mathcal{A}$ is given access to the oracle $\mathsf{MAC}(k,\cdot)$. In the end, $\mathcal{A}$ outputs a message-tag pair (m,τ). Its advantage measure $\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{MAC}}(\mathcal{A})$ is defined as the probability that $\mathsf{MAC}(m) = \tau$ and m was not queried to the $\mathsf{MAC}(k,\cdot)$ oracle.

MAC is *existentially unforgeable under chosen message attack (EUF-CMA)* if for any PPT adversary $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{MAC}}(\mathcal{A}) = \mathsf{negl}(\log|\mathcal{K}|)$.

2.5 Authenticated Encryption

We briefly recall the authenticated encryption scheme and its security notions and refer to [43] for detailed definitions.

For an authenticated encryption scheme $\mathcal{ES} = (\mathcal{K}, \mathsf{Enc}, \mathsf{Dec})$, its decryption should fail when the ciphertext has not been properly generated under the appropriate key. This will thus provide a kind of key confirmation, as usually done to achieve explicit authentication. However, some critical data will have to be sent, hence a simple MAC would not be enough, privacy of the content is important too.

There are two security notions for an authenticated encryption scheme. The *semantic security*, also known as *indistinguishability under chosen plaintext attack (IND-CPA)*, prevents any information being leaked about the plaintexts, while the *integrity of ciphertexts (INT-CTXT)* essentially guarantees no valid ciphertext can be produced without the key. Let $\mathbf{Adv}_{\mathcal{ES}}^{\mathsf{ind\text{-}cpa}}(\mathcal{A})$ and $\mathbf{Adv}_{\mathcal{ES}}^{\mathsf{int\text{-}ctxt}}(\mathcal{A})$ respectively denote the adversarial advantages for the above notions. For simplicity, let $\mathbf{Adv}_{\mathcal{ES}}^{\mathsf{authenc}}(\mathcal{A}) = \max\{\mathbf{Adv}_{\mathcal{ES}}^{\mathsf{ind\text{-}cpa}}(\mathcal{A}), \mathbf{Adv}_{\mathcal{ES}}^{\mathsf{int\text{-}ctxt}}(\mathcal{A})\}$.

One simple way to achieve secure authenticated encryption is by using a generic Encrypt-then-MAC approach [43] or by using a dedicated scheme such as OCB [44].

2.6 Stateful Authenticated Encryption with Associated Data

We follow [45, 46] in extending the stateful authenticated encyption notion of Bellare *et al.* [47] to capture a hierarchy of stateful AEAD security notions based on different authentication levels. The following definitions are the same as [46], except that we exclude the length-hiding property proposed by Paterson *et al.* [48] for conciseness.

Syntax. A stateful AEAD scheme sAEAD is a three-tuple $(\mathsf{sGen}, \mathsf{sEnc}, \mathsf{sDec})$ associated with a key space $\mathcal{K} = \{0,1\}^\lambda$, a message space $\mathcal{M} \subseteq \{0,1\}^*$, an associated data space $\mathcal{AD} \subseteq \{0,1\}^*$, and a state space $\mathcal{ST} \subseteq \{0,1\}^*$. sGen is a probabilistic algorithm that samples a random key from $\mathcal{K}$ and initializes the encryption and decryption states $st_e, st_d \in \mathcal{ST}$. sEnc is a probabilistic encryption algorithm that takes as input $k \in \mathcal{K}, ad \in \mathcal{AD}, m \in \mathcal{M}$ and st_e and outputs a ciphertext $ct \in \{0,1\}^*$ with an updated st_e. sDec is a deterministic decryption algorithm that takes as input $k \in \mathcal{K}, ad \in \mathcal{AD}, ct \in \{0,1\}^*$ and st_d and outputs $m \in \mathcal{M} \cup \{\perp\}$ with an updated

st_d. *Correctness* requires that, for any $k \in \mathcal{K}, st_e = st_e^0, st_d = st_d^0$ sampled or initialized by sGen and any sequence of encryptions $\{(ct_{i+1}, st_e^{i+1}) \xleftarrow{\$} \mathsf{sEnc}(k, ad_i, m_i, st_e^i)\}_{i \geq 0}$, the sequence of decryptions $\{(m'_{i+1}, st_d^{i+1}) \leftarrow \mathsf{sDec}(k, ad, \mathsf{Enc}(k, ad_i, ct_i, st_d^i))\}_{i \geq 0}$ satisfies $m_i = m'_i, i \geq 0$.

Security. Consider the following experiment with an authentication level $al \in [4]$. In the beginning, run sGen to generate a key k and initialize st_e, st_d. Sample $b \xleftarrow{\$} \{0, 1\}$ and set $(u, v, \mathtt{outofsync}) \leftarrow (0, 0, 0)$. Then, the adversary $\mathcal{A}$ is given access to the following oracles:

```
encrypt($ad, m_0, m_1$):
  1: $u \leftarrow u + 1$, $(sent.ct_u, st'_e) \xleftarrow{\$} \mathsf{sEnc}(k, ad, m_b, st_e)$
  2: $(sent.ad_u, st_e) \leftarrow (ad, st'_e)$, return $sent.ct_u$
decrypt($ad, ct$):
  1: if $b = 0$, return $\perp$
  2: $v \leftarrow v + 1$, $(m, st'_d) \leftarrow \mathsf{sDec}(k, ad, ct, st_d)$
  3: $(rcvd.ad_v, st_d) \leftarrow (ad, st'_d)$
  4: if $(al = 4) \land \mathsf{cond}_4$ or $(al \leq 3) \land (m \neq \perp) \land \mathsf{cond}_{al}$,[1]
         set $\mathtt{outofsync} \leftarrow 1$
  5: if $\mathtt{outofsync} = 1$, return $m$, otherwise, return $\perp$
```

In the end, $\mathcal{A}$ outputs a bit b'. The stateful AEAD scheme sAEAD is *secure* with authentication level al if and only if $\mathbf{Adv}_{\mathsf{sAEAD}}^{\mathsf{aead}\text{-}al}(\mathcal{A}) = |2\Pr[b = b'] - 1|$ is negligible in λ for any PPT adversary $\mathcal{A}$.

2.7 Collision-Resistant Hash Function Family

Consider a function family $H = \{h_k : \mathcal{D}_k \to \mathcal{R}_k\}_k$ generated by some algorithm $G(1^\lambda)$, where $\forall k \xleftarrow{\$} G(1^\lambda), |\mathcal{D}_k| > |\mathcal{R}_k|$ and computing h_k on any input is efficient given k. In the security experiment, an adversary $\mathcal{A}$ takes as input 1^λ and a random $k \xleftarrow{\$} G(1^\lambda)$, then

[1] Authentication conditions cond_{al} are defined in the same way as in the msACCE-std Decrypt query.

outputs two messages $(x_1, x_2) \in \mathcal{D}_k$. Its advantage measure $\mathbf{Adv}_H^{\mathsf{coll}}(\mathcal{A})$ is defined as the probability that $x_1 \neq x_2$ and $h_k(x_1) = h_k(x_2)$.

H is *collision-resistant* if for any PPT adversary $\mathcal{A}$, $\mathbf{Adv}_H^{\mathsf{coll}}(\mathcal{A}) = \mathsf{negl}(\lambda)$. Note that in practice H is typically a single hash function instead of a function family.

2.8 Signature Scheme

A signature scheme Sig consists of three efficient algorithms $(\mathsf{Kg}, \mathsf{Sign}, \mathsf{Ver})$:

- Kg: takes as input the security parameter 1^λ and outputs a pair of keys: a public verification key pk and a private signing key sk.
- Sign: takes as input a signing key sk and a message m, then outputs a signature σ.
- Ver: takes as input a verification key pk, a message m, and a signature σ, then outputs a bit b indicating if the signature is valid.

Correctness requires that for any key pair $(pk, sk) \xleftarrow{\$} \mathsf{Kg}(1^\lambda)$ and any message m, $\mathsf{Ver}(pk, m, \mathsf{Sign}(sk, m)) = 1$.

For security, consider the following security experiment associated with an adversary $\mathcal{A}$. In the beginning, run $(pk, sk) \xleftarrow{\$} \mathsf{Kg}(1^\lambda)$. Then, $\mathcal{A}$ is given pk and access to the oracle $\mathsf{Sign}(sk, \cdot)$. In the end, $\mathcal{A}$ outputs a message-signature pair (m, σ). Its advantage measure $\mathbf{Adv}_{\mathsf{Sig}}^{\mathsf{euf\text{-}cma}}(\mathcal{A})$ is defined as the probability that $\mathsf{Ver}(pk, m, \sigma) = 1$ and m was not queried to the $\mathsf{Sign}(sk, \cdot)$ oracle.

Sig is *existentially unforgeable under chosen message attack (EUF-CMA)* if for any PPT adversary $\mathcal{A}$, $\mathbf{Adv}_{\mathsf{Sig}}^{\mathsf{euf\text{-}cma}}(\mathcal{A}) = \mathsf{negl}(\lambda)$.

2.9 The Diffie-Hellman Assumptions

Consider a cyclic group $\mathbb{G} = \langle g \rangle$ of prime order q associated with the security parameter λ.

The *computational Diffie-Hellman (CDH)* assumption states that it is computationally infeasible to compute g^{ab} given $\mathbb{G}, g, g^a, g^b$ for random $a, b \xleftarrow{\$} \mathbb{Z}_q$. That is, let $\mathbf{Adv}_{\mathbb{G}}^{\mathsf{cdh}}(\mathcal{A})$ denote the probability that an adversary $\mathcal{A}$ outputs g^{ab}, then we have $\mathbf{Adv}_{\mathbb{G}}^{\mathsf{cdh}}(\mathcal{A}) = \mathsf{negl}(\lambda)$ for any PPT adversary $\mathcal{A}$. On the other hand, the *decisional Diffie-Hellman (DDH)* assumption states that (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) are computationally indistinguishable, where c is chosen uniformly at random from $\mathbb{Z}_q$. Similarly, let $\mathbf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(\mathcal{A})$ denote the associated adversarial advantage.

For the *strong CDH (SCDH)* assumption [49], an adversary $\mathcal{A}$ is additionally granted oracle access to $\mathcal{O}_a(\cdot, \cdot)$, which takes any group elements $Y, Z \in \mathbb{G}$ as input and checks if $Y^a = Z$. Let $\mathbf{Adv}_{\mathbb{G}}^{\mathsf{scdh}}(\mathcal{A})$ denote the probability that $\mathcal{A}$ outputs g^{ab}. The SCDH assumption states that for any PPT adversary $\mathcal{A}$, $\mathbf{Adv}_{\mathbb{G}}^{\mathsf{scdh}}(\mathcal{A}) = \mathsf{negl}(\lambda)$.

2.10 Password-Authenticated Key Exchange

A *Password-Authenticated Key Exchange (PAKE)* protocol is an interactive protocol between two parties (sometimes referred to as a client and a server). PAKE protocols allow them to establish an high-entropy session key over an insecure channel using only a shared low-entropy, human-memorizable password.

Since only the shared low-entropy password is used for authentication, an adversary has non-negligible chance of successfully impersonating one of the parties by guessing their shared password. Such an impersonation attack is called an *online* dictionary attack because the adversary cannot mount this attack by itself (referred to as *offline* dictionary attack) but needs to interact with an "online" party to verify its guess. Note that an offline attack, if possible, is disastrous to a PAKE protocol due to the low entropy of the password. Informally, a secure PAKE protocol guarantees that for any efficient adversary an exhaustive online dictionary attack is the best strategy to break the protocol.

We consider the security model for PAKE in the universal composability (UC) framework [51], which makes no assumption on the distribution of passwords. The UC security

The functionality $\mathcal{F}_{\mathsf{PAKE}}$ is parameterized by a security parameter k. It interacts with an adversary $\mathcal{S}$ and a set of parties P_1,...,P_n via the following queries:

Upon receiving a query $(\texttt{NewSession}, sid, P_i, P_j, pw, role)$ **from party** P_i**:**

Send $(\texttt{NewSession}, sid, P_i, P_j)$ to $\mathcal{S}$.

If this is the first `NewSession` query, or if this is the second `NewSession` query and there is a record (P_j, P_i, pw'), then record (P_i, P_j, pw) and mark this record `fresh`.

Upon receiving a query $(\texttt{TestPwd}, sid, P_i, pw')$ **from the adversary** $\mathcal{S}$**:**

If there is a record of the form (P_i, P_j, pw) which is `fresh`, then do:

- If $pw = pw'$, mark the record `compromised` and reply to $\mathcal{S}$ with "correct guess".
- If $pw \neq pw'$, mark the record `interrupted` and reply with "wrong guess".

Upon receiving a query $(\texttt{NewKey}, sid, P_i, sk)$ **from the adversary** $\mathcal{S}$**:**

If there is a record of the form (P_i, P_j, pw) and this is the first `NewKey` query for P_i, then do:

- If this record is `compromised`, or either P_i or P_j is corrupted, then output (sid, sk) to P_i.
- If this record is `fresh`, and there is a record (P_j, P_i, pw') with $pw = pw'$, and a key sk' was sent to P_j, and (P_j, P_i, pw') was `fresh` at the time, then output (sid, sk') to P_i.
- In any other case, pick a random key sk' of length k and send (sid, sk') to P_i.

Either way, mark the record (P_i, P_j, pw) as `completed`.

Figure 2.1: Functionality $\mathcal{F}_{\mathsf{PAKE}}$ [50]

of a PAKE protocol is defined with respect to its ideal functionality, for which we consider both the basic functionality $\mathcal{F}_{\mathsf{PAKE}}$ (see Figure 2.1) from [50] and the functionality $\mathcal{F}^{\mathsf{CA}}_{\mathsf{PAKE}}$ defined in [52] that captures client authentication. Both functionalities can be efficiently instantiated with (variants of) the classical EKE [17] protocol that uses the shared password to encrypt a Diffie-Hellman key exchange. In particular, [52] shows that a variant [53] of the EKE protocol securely realizes $\hat{\mathcal{F}}^{\mathsf{CA}}_{\mathsf{PAKE}}$ (the implicitly defined multi-session extension [54] of $\mathcal{F}^{\mathsf{CA}}_{\mathsf{PAKE}}$), under the computational Diffie-Hellman assumption in the random oracle and ideal cipher models. There also exist other less efficient secure PAKE constructions in the standard model, e.g., the protocol proposed in [50].

In our analysis, we use $\mathbf{Adv}^{\mathsf{pake}}_{\mathsf{PAKE}}(\mathcal{S}, \mathcal{A}, \mathcal{Z})$ to denote the advantage of the environment $\mathcal{Z}$ distinguishing between the ideal world with the ideal functionality ($\hat{\mathcal{F}}_{\mathsf{PAKE}}$ or $\hat{\mathcal{F}}^{\mathsf{CA}}_{\mathsf{PAKE}}$) and the simulator $\mathcal{S}$ and the real world with the PAKE protocol PAKE and the adversary $\mathcal{A}$. We also assume the "black-box" simulator exists for PAKE and denote it by $\mathcal{S}_{\mathsf{PAKE}}$.

Note that PAKE may correspond to a PAKE instantiation that realize either $\hat{\mathcal{F}}_{\mathsf{PAKE}}$ or $\hat{\mathcal{F}}^{\mathsf{CA}}_{\mathsf{PAKE}}$. In particular, our first work presented in Chapter 3 uses $\hat{\mathcal{F}}_{\mathsf{PAKE}}$ while our final work presented in Chapter 5 uses $\hat{\mathcal{F}}_{\mathsf{PAKE}}^{\mathsf{CA}}$.

CHAPTER 3
HUMAN AUTHENTICATED KEY EXCHANGE

3.1 Introduction

As motivated in Subsection 1.2.1, our goal in this work is to provide the first solution to the following problem: if a user's terminal has been fully compromised, can we still protect his (past and future) sessions from other trusted terminals, even though the *same* long-term secret is used?

3.1.1 Related Work

As we mentioned, there are numerous results about PAKE, from the seminal work of Bellovin and Merritt [17, 43], but they offer no practical solutions for strong corruptions (to protect future sessions).

Matsumoto and Imai [55] proposed the first scheme to deal with human identification through insecure channels (and via untrusted machines), which has been improved by the follow-up works [56, 57, 58]. However, their schemes are only secure given very few login sessions or require the human to memorize a long bit string. Later, Blocki *et al.* [18] improved this line of work with a much more secure and practical construction. Dziembowski [59] also considers the problem of human-based key-exchange, but in a setting where both parties are human and his scheme is only secure against a machine adversary unable to solve CAPTCHAs.

There is a long sequence of papers [60, 61, 62, 63, 64, 65, 66] following the work by Hopper and Blum [67] offering protocols for the same problem of secure human identification over insecure channels, whose security is based on the Learning Parity with Noise (LPN) problem. They require too much computation work for human users. Personal devices generating one-time passwords have been commercially available for years [68], motivating IETF to standardize their constructions and use in many protocols [69, 70, 71, 72, 73]. The work [73] is particularly relevant to our device-assisted constructions and it defined a time-based one-time password algorithm based on HMAC [72]. Interestingly, while dedicated token generators are the most secure, software applications running on mobile phones are now commonly used [74].

Some papers explore PAKE schemes with one-time passwords. Paterson and Stebila [75] defined a security model for one-time PAKE, explicitly considering the compromise of past (and future) one-time passwords, but still recovering the security after a compromise (which is still not strong corruption, i.e., not all passwords are corrupted), thanks to the ephemeral property of the one-time password and its change over the time. However, their construction is generic and uses PAKE as a black-box. It thus cannot be more efficient than a PAKE, as opposed to ours. They also mentioned the possibility of using a secure token to generate the one-time passwords and then running one-time-PAKE on it, but they did not provide an explicit protocol or security analysis.

3.1.2 Our Contributions

To make our basic ideas described in Subsection 1.2.1 "work", numerous problems need to be resolved to finalize the solutions. We discuss these after we describe our security model.

Protocol and Security Definitions. The novelty behind our definitions is the unavoidable incorporation of a human player. We define a *Human Authenticated Key Exchange (HAKE)* protocol as an interactive protocol between a human user U and a server S, via a terminal T. The server can only directly communicate with the terminal, and the user can only directly communicate with the terminal. In addition, the messages sent by the terminal to the user must be *human-readable*, the messages sent by the user to the terminal must be *human-writable*, and the long-term secret of the user must be *human-memorizable*, unless an additional device is used for computing the ephemeral secrets.

Our security model is a non-trivial extension of the security model for PAKE protocols by Bellare, Pointcheval, and Rogaway [43], later called BPR, as we take into account really strong corruptions and model human computations/interactions. As already mentioned, the goal of a HAKE protocol is to ensure that a human user sharing the long-term secret with a server can establish a secure channel with the server, in presence of a very strong attacker. Our model takes into account various types of attacks possible in practice. As usual, to model a network compromise (e.g., taking advantage of an insecure Wi-Fi), we allow the adversary to control the messages parties exchange. The attacker can read and modify the communication between a server and a terminal. However, we assume that the channel between the human user and the *honest* terminal is secure, since this is a direct communication from the keyboard and the screen. At least, it is authentic and private, unless the terminal is compromised.

We thus also have to model malicious terminals and this, in fact, is the gist of our work. Even though taking the full control of a computer is an extremely hard job, compromising its parts, such as a browser, is very common. And such compromises can be of various strengths. Using a keylogger, screen capture or similar malware the adversary can learn the terminal's inputs/outputs. The attacker may also learn some random coins or intermediate values from the internal state of the compromised computer. This also models human "over-the-shoulder" attacks. Even though such compromise can be referred to as honest-but-curious, the existing protocols, such as PAKE, do not offer protection against it. The existing protocols only protect against *weak* corruptions, where the attacker just learns the session key (with reveal-queries). This models the misuse of the session key, rather than the terminal compromise. Even if security models for PAKE allow the attacker to learn the long-term secrets [43] (with corrupt-queries), or the internal states (in the UC framework [51]), this is only to model forward secrecy, and so the security of past sessions, but nothing is guaranteed anymore for future sessions.

In our model, we let the adversary compromise terminals and learn all their inputs and the internal state. Moreover, we consider an even more powerful adversary, who takes full control of the terminal's browser and can display outputs of its choice to be shown to the human user. Hence, the adversary can interact with the human user, but is never given the long-term secret key memorized by the human user (or stored on his secondary device).

The security goals are, to the most part, the standard privacy and authentication for key exchange protocols: we want to make sure that an attacker cannot learn any information about the session key nor make a party agree on a session key without the other party completing the protocol. Of course, if a terminal is compromised, it is unreasonable to expect security of the current session. But this should not compromise security of other sessions (past or future), even involving the same user.

HAKE Protocol: Generic Constructions and Instantiations. Let us assume we have a *Human-Compatible (HC)* function family F (we will discuss it in more detail shortly). Let the server pick a random challenge x (or increment a counter) and display it to the human user via the user's terminal. The user can compute (in his head or using a device) and enter the response $r = F_K(x)$, where K is the long-term secret shared between the user and the server. The server can compute r on its end the same way. Then, the terminal and the server execute a PAKE protocol on r (i.e., the response r plays the role of the password in PAKE), and thus agree on a session key. Even though human-computable responses may have low entropy, PAKE ensures security against off-line dictionary attacks, which guarantees no information leakage about the ephemeral secret r in passive sessions, and even in active sessions, excepted possibly the exclusion of one candidate per session. If the attacker compromises the terminal, a suitable "unforgeability" property of F would prevent the adversary from breaking security of other sessions.

But still, this protocol is not secure under our definition. An attacker, who learns an ephemeral secret $r = F_K(x)$ for a given challenge x can later use it to successfully impersonate the server, by forcing the same challenge. To prevent such replay attacks, we let the terminal and the server to jointly pick a challenge using a coin-flipping protocol, that we implement using a commitment scheme with specific properties. This is our first proposal, which we call the *Basic HAKE*: using a coin-flipping, we avoid replay attacks, and with a suitable unforgeability property on the function family F we can guarantee the security of the global process.

However, a malicious terminal can still ask specific (not necessarily random) challenges to the human user, while impersonating the server, and the user has no way to detect such a malicious behavior. Therefore security of the Basic HAKE requires that the HC function unforgeability holds even in presence of multiple adaptive challenges. This may be too strong of a requirement in practice. We thereafter enhance *Basic HAKE* and propose the *Confirmed HAKE* protocol, which allows parties to detect potential bad behaviors, in order to react appropriately, and thus the construction tolerates weaker HC function families. Requirements on HC function families then become more compatible with functions that can be evaluated by human being without external help. The Confirmed HAKE also provides explicit authentication of the parties.

Finally, we consider the case of *device-assisted* protocols: with an additional device, one can implement more complex computations and use stronger HC function families. This leads to less critical ephemeral secrets: leaking information about several $(x, F_K(x))$ pairs might not endanger the long-term secret K. This allows us to rely on a weaker variant of PAKE, and hence get device-assisted HAKE protocols that are more efficient.

Human-Compatible Function Family. We now turn our attention to the inner part of the construction, the human-compatible function family. Security-wise, the adversary should be able to see multiple challenge-response pairs, among which some of the challenges could be chosen by the attacker (adaptive queries vs. non-adaptive queries). This is because the attacker, who compromises a terminal, can eavesdrop on the communication with the human user. And an active attacker who took control of the terminal can impersonate the server and ask the user to answer maliciously chosen challenges. But still, the adversary should not be able to forge a valid response for a new random challenge, so that future sessions remain safe.

Finding such a function family would be easy if we did not have the human-computability restrictions. We survey some works on secure human-based computation later, but they are not directly suitable for us. Luckily, a recent paper by Blocki, Blum, Datta, and Vempala [18] (almost) provides a solution. They proposed a way for a human user to authenticate to a computer that does not offer privacy (honest-but-curious). Such a computer stores a set of challenges and the user authenticates by providing a response to a random challenge. In their concrete construction, a challenge is a set of images, the secret user memorizes is a correspondence between images and numbers and the response is some basic function using addition of the digits (modulo 10). The authors provide experimental evidence that their scheme can be used by a human user. Namely, the secret can be memorized and the response can be computed within reasonable time by an average human user. The authors also propose a tool to help secret memorization. While the usability of their solution is not perfect, it is definitely a start and further research will hopefully yield protocols with better usability.

Security-wise, the authors prove that recovering the user's long-term secret from a number (below a certain bound) of random challenge-response pairs (non-adaptive queries) is equivalent to solving the random planted constraint satisfiability problem, and they state a conjecture about security of the latter. To support the conjecture, the authors prove the hardness of the problem for any *statistical* attacker, extending the results of [76]. Finally, it is proven that forging a response for a random challenge is equivalent to recovering the secret. The bound on the number of revealed challenge-response pairs corresponds to the maximum number of logins a user can execute, without endangering future sessions.

The construction and security results from [18] are very useful for our work, but we cannot use them as is. The problem is that it is not known whether security of their scheme holds when the attacker can see responses to maliciously chosen challenges (adaptive queries). We extend their analysis and prove a second conjecture that the unforgeability of their HC function family still holds if the adversary can make very few adaptive queries. Our Confirmed HAKE is designed to rely on such HC functions (whose security can tolerate very few adaptive queries): after the PAKE completion using the first response, the human user selects a random challenge and enters it into the terminal, who encrypts the challenge under the recently established session key and forwards the result to the server. The server decrypts, computes the response, and sends it, also encrypted to the terminal. The terminal decrypts and displays the response and the human user verifies it. If verification fails, the user needs to take measures against suspected terminal infection and possibly abort the long-term secret. Encrypting the terminal-server communication here is needed for authenticity in case of an honest terminal, to prevent a network adversary to ask the server maliciously chosen challenges. We show that this extended protocol limits the number of responses the attacker infecting the terminal can obtain for malicious challenges of its choice (in that case, the adversary will not be able to make the user pass the connection confirmation step). We argue that this addition, while adds a little bit more work for the human user, does not violate human computability for our instantiation, i.e., that the user can select a random challenge and verify the response. Furthermore, we show that the Confirmed HAKE provides explicit authentication assuming that the encryption scheme is secure authenticated encryption.

Our formal analysis on the HC function [18] demands a stronger conjecture stating *one-more* unforgeability. This is similar to the analysis of blind signatures that relies on the one-more unforgeability of RSA [77], but we consider a sequential version of the one-more security definition, that is weaker than the original one.

We want to note that, unlike [18], in our analysis, the bound on the number of challenge-response pairs the attacker can see does not correspond to the total number of logins, but only to the number of logins via compromised terminals, which is much more practical. This is because PAKE guarantees security against network attackers when end-points are secure: responses remain completely hidden to external players.

Unfortunately, it is not clear how to extend the results of [18] to expect resistance to many adaptive queries (so that we could have a simpler protocol without the confirmation step). The only possibility is the use of a pseudo-random function: after many adaptive queries, the response to a new challenge is still random-looking to any adversary. But for such functions, one needs additional help, hence our *device-assisted* scenario. One important advantage of such a stronger HC function family (tolerating many adaptive challenges, and thus also many non-adaptive challenges) is that responses are ephemeral secrets used once for authentication, but that can be revealed after use: as a consequence, a weaker variant of PAKE is enough, since resistance to off-line dictionary attacks is not required any more. We can expect more efficient constructions. Hence is our first construction in the device-assisted context. But to limit interactions with the device and avoid collisions on the inputs, we thereafter adopt a time-based challenge: $r = F_K(t)$, with an increasing counter t, based on an internal clock. While one cannot guarantee perfect synchronization between the device and the server, we can tolerate a slight time-shift since we anyway use timeframes that are long enough for the human to enter the response read on the device (e.g., 30 seconds or 1 minute).

To summarize, we propose the first user authenticated key exchange protocols which can tolerate corrupted terminals: if a user happens to log in to a server from a terminal that has been fully compromised, then the other past and future user sessions initiated from honest terminals stay secure. We formalize the security model for Human Authenticated Key Exchange (HAKE) protocols and propose both generic and device-assisted constructions based on HC function families, PAKE, commitment scheme, and authenticated encryption. We analyze the security of our HAKE protocols and discuss their instantiations. We hope our work will promote further developments in the area of human-oriented cryptography.

3.2 HAKE Syntax and Security Model

In this section, we define the syntax and security model for a *Human Authenticated Key Exchange (HAKE)* protocol.

3.2.1 Protocol Syntax

HAKE is an interactive protocol between a user and a server, via a terminal, which can be viewed as an extension to an *Authenticated Key Exchange (AKE)* protocol [43] by separating an AKE client into a user and a terminal and capturing human interactions.

Human-Compatible Communication. Here we present several notions that our protocol definition will use. Since it is hard to formalize human computational abilities, our definitions are not mathematically precise.

We say a message is *human-readable* if this is a short sequence of ASCII symbols, or images; *human-writable* if this is a short sequence of ASCII symbols[1]; *human-memorizable* if this is simple enough to be memorized by an average human, e.g., a simple arithmetic rule like "plus 3 modulo 10". A function is *human-computable* if an average human can evaluate it without help of additional resources other than his head, e.g., simple additions modulo 10. A set is *human-sampleable* if an average human can choose a message from the set at random according to the appropriate distribution without help of additional resources other than his head.

HAKE Syntax. We now formally describe a HAKE protocol.

Definition 1 (HAKE Protocol)**.** *A* human authenticated key exchange *protocol is an interactive protocol between a human user U and a server S, via a terminal T. It consists of two algorithms:*

- *A long-term key generation algorithm $\mathcal{LKG}$ which takes as input the security parameter and outputs a long-term key.*

[1]It is also possible to incorporate mouse clicks into that, but we do not deal with it for simplicity.

- *An interactive key-exchange algorithm $\mathcal{KE}$ which is ran between U, T, and S. At the beginning, only U and S take as input the same long-term secret key and, at the end, T and S each outputs a session key sk_T and sk_S respectively. In case of additional explicit authentication, U and/or S may either* accept *or* reject *the connection.*

The above algorithms must satisfy the following constraints:

- *S can only communicate with T;*
- *U can only communicate with T, and*
  - *the message sent by T to U must be* human-readable*, and*
  - *the message sent by U to T must be* human-writable*;*
- *The long term secret and the state of U, if any, must be* human-memorizable *for the duration necessary.*

Correctness requires that for every security parameter and for every long-term key output by $\mathcal{LKG}$, in any execution of $\mathcal{KE}$, U and S both accept the connection (in case of explicit authentication), T and S complete the protocol with the same session key ($sk_T = sk_S$).

3.2.2 Security Model

As already mentioned, the goal of a HAKE protocol is to ensure that a human user sharing the long-term secret with a server can help a terminal to establish a secure channel with the server, in presence of a very powerful attacker, including strong corruptions of terminals.

Let $\mathcal{P}$ denote the set of all participants. As usual, to model multiple and possibly concurrent (except for the human users) sessions we consider oracles π_P^i, where $i \in \mathbb{N}_+$ and $P \in \mathcal{P}$. For human oracles, sessions can only be sequential, and not concurrent, meaning that humans are not allowed to run several sessions concurrently (a new session starts after the previous one ends). This is a reasonable assumption for human users. We note that since terminals do not store long-term secrets and do not preserve state between sessions, multiple terminal oracles model both multiple sessions ran from the same or different terminals. Also, we assume the server cannot be compromised.

Hence, without loss of generality, in the following we consider multiple human users $\{U_\ell\}_\ell$ with different long-term secrets, *one* terminal T, and *one* server S that has all the user long-term secrets. For all of them, multiple instances will model the multiple sessions (either sequential for U_ℓ, or possibly concurrent for T and S). However, while the server can concurrently run several sessions, we will also limit it to one session at a time with each user: the server will not start a new session with a user until it finishes the previous session with the same user.

Because of our specific context with a human user, there is a direct communication link between the user and the terminal, and so we can assume that the channels between instances $\pi^i_{U_\ell}$ and π^j_T are authenticated and even private (unless the terminal oracle is *compromised*, as defined below), whereas the communication between the terminal and the server is over the internet, and so the channels between instances π^j_T and π^k_S are neither authenticated nor private.

Security Experiments. We consider the following security experiments associated with a given HAKE protocol and an adversary $\mathcal{A}$, to define the two classical security notions for authenticated key exchange: privacy (or semantic security of the session key) and authentication. In these experiments, the adversary $\mathcal{A}$ can make the following queries:

• $\mathsf{Compromise}(j, \ell)$, where $j, \ell \in \mathbb{N}$ – As the result of this query, the terminal oracle π^j_T is considered to be *compromised*, and the adversary gets its internal state, i.e. the random tape, temporary variables, etc. If the terminal oracle π^j_T is not linked yet to a user, it is linked to user U_ℓ with the user oracle $\pi^i_{U_\ell}$ for a new index i, otherwise ℓ is ignored.

• $\mathsf{Infect}(j)$, where $j \in \mathbb{N}$ – As the result of this query, the terminal oracle π^j_T is considered to be *infected*. Without loss of generality, we limit this query to *compromised* terminals only.

• $\mathsf{SendTerm}(j, M)$, where $j \in \mathbb{N}$ and $M \in \{0,1\}^* \cup \{\mathsf{Start}(\ell)\}$ – This sends message M to π_T^j. A specific $\mathsf{Start}(\ell)$ message asks the terminal to initiate a session, to be done with a user oracle $\pi_{U_\ell}^i$ for a new index i. But only if the terminal oracle π_T^j is not linked yet to a user, otherwise ℓ is ignored. To compute its response to $\mathcal{A}$, π_T^j may internally talk to its linked human oracle according to the protocol. In addition, if π_T^j is *compromised*, it will additionally return to $\mathcal{A}$ the messages exchanged with its linked human oracle[2].

• $\mathsf{SendServ}(k, M)$, where $k \in \mathbb{N}$ and $M \in \{0,1\}^*$ – This sends message M to oracle π_S^k. The oracle computes the response according to the corresponding algorithm and sends the reply to $\mathcal{A}$.

• $\mathsf{SendHum}(j, M)$ where $j \in \mathbb{N}$ and $M \in \{0,1\}^*$ (and human-readable) – This sends a message to the π_T^j linked human oracle $\pi_{U_\ell}^i$ on behalf of π_T^j. This is allowed only if the terminal π_T^j is *infected* (and thus *compromised*, which implies the existence of a partenered human oracle). The oracle computes the response according to the corresponding algorithm and sends the reply to $\mathcal{A}$.

• $\mathsf{Test}(j, P)$, where $j \in \mathbb{N}$ and $P \in \{T\} \cup \{S\}$ – If sk_P has been output by π_P^j, then one looks at the internal bit b (flipped once for all at the beginning of the privacy experiment, while $b = 1$ in the authentication experiment). If $b = 1$, then $\mathcal{A}$ gets the real session key sk_P, otherwise it gets a uniformly random session key. This query is only allowed if π_P^j is fresh (defined below).

In the privacy experiment, after having adaptively asked several of these oracle queries, the adversary $\mathcal{A}$ outputs a bit b' (a guess on the bit b involved in the Test queries). The intuition is that the adversary should not be able to distinguish the real session keys from independent random strings. While in the authentication experiment, the goal of the adversary is to make an honest party to successfully complete the protocol execution thinking it "built a secure session" with the right party, whereas that is not the case. In order to formally define the goals and the advantages of the adversary, we present the notions of

[2]The messages to the human oracle can be already known to the adversary as they are a function of the oracle's random tape. But we give the adversary the whole communication for convenience.

partnering and freshness, as well as the flags accept and $\mathsf{terminate}$.

Flags. In order to model authentication, we follow BPR [43], who defined two flags: accept essentially means that a party has all the material to compute the session key while $\mathsf{terminate}$ means that a party thinks that it completes the protocol execution thinking it communicates with the expected other party (a human user in our case). These two flags are initially set to `False`, and they are explicitly set to `True` in the description of the protocol. Note that in Definition 1 U and/or S *accept* if and only if in the end the $\mathsf{terminate}$ flag is set to `True`, otherwise, U and/or S *reject*.

Partnering. Whereas $\pi^i_{U_\ell}$ and π^j_T are declared as *linked* at the initialization of the communication because of the authenticated channels between users and the terminal, partnering between $\pi^i_{U_\ell}$ and π^k_S is *a posteriori*: they are indeed declared *partners* in the end of the protocol execution if they use the same long-term key and both accept. Then we define partnering between π^j_T and π^k_S, by saying that they are declared *partners* if π^k_S and U^i_ℓ are partners and U^i_ℓ is linked to π^j_T.

Freshness. Informally, the freshness denotes oracles that hold sessions keys that are not trivially known to the adversary. For $P \in \{T\} \cup \{S\}$, the oracle π^j_P is *fresh*, if no Test query has been asked to π^j_P nor its partner, and none of π^j_P or its partner have been *compromised* (π^j_T is fresh if it has not been compromised, and π^k_S is fresh if the terminal *linked* to the partner human user has not been compromised.)

Security Notions. In the privacy security game, the goal of the adversary is to guess the bit b involved in the Test queries. Then we measure the success of an adversary $\mathcal{A}$, that outputs a bit b', by $\mathbf{Adv}^{\mathsf{priv}}_{\mathsf{HAKE}}(\mathcal{A}) = |2 \cdot \Pr[b' = b] - 1|$. This notion implies *implicit* authentication, which essentially means that no one else than the expected partners share the session key material.

For *explicit* authentication, we define the authentication security game: the goal of the adversary is essentially to make a player terminate (flag $\mathsf{terminate}$ set to true) without an accepting partner (flag accept set to true). But in our case with *compromised* or even *infected* terminals, this is a bit more complex than usual. We thus split the authentication security in two parts:

- Server-authentication: a user oracle should not successfully terminate a session if there is not exactly one partner server oracle that has accepted. Then, we denote $\mathbf{Adv}^{\mathsf{s\text{-}auth}}_{\mathsf{HAKE}}(\mathcal{A})$ the probability the adversary $\mathcal{A}$ makes such a bad event happens;
- User-authentication: a server oracle should not successfully terminate a session if there is not exactly one partner user oracle that has accepted. Then, we denote $\mathbf{Adv}^{\mathsf{u\text{-}auth}}_{\mathsf{HAKE}}(\mathcal{A})$ the probability the adversary $\mathcal{A}$ makes such a bad event happens.

Eventually, we define $\mathbf{Adv}^{\mathsf{auth}}_{\mathsf{HAKE}}(\mathcal{A}) = \max\{\mathbf{Adv}^{\mathsf{s\text{-}auth}}_{\mathsf{HAKE}}(\mathcal{A}), \mathbf{Adv}^{\mathsf{u\text{-}auth}}_{\mathsf{HAKE}}(\mathcal{A})\}$.

For simplicity, in the following we only consider a single user for the above security notions, which can be easily extended to the multi-user case via a union bound.

Passive Sessions. We now define a new notion of *passive session*, which extends the Execute queries in the standard BPR model [43]. Recall that Execute queries allow the adversary to get full transcripts of communication between honest parties. Even though the same can be achieved via Send queries, in the security analyses it is useful to count the number of observed honest sessions and the number of maliciously altered sessions separately. In addition, we will not limit to full sessions: the adversary can stop forwarding honest flows, making the session abort. Then, there can be *passive full/partial-sessions*:

Definition 2 (Passive Session)**.** *A (full or partial) session between oracles π_T^j and π_S^k is called* passive*, if the messages of all queries* $\mathsf{SendTerm}(j, \cdot)$ *or* $\mathsf{SendServ}(k, \cdot)$ *are either* $\mathsf{Start}(\cdot)$ *or themselves an output of one of these two queries type. If flows are numbered, this also implies that the actual order of flows between T and S has not been modified. If all the outputs have been forwarded as inputs, this is a* passive full-session*, otherwise this is a* passive partial-session*.*

Sessions that are not passive are called *active*, since the adversary altered something in the honest execution.

We believe this notion is stronger than the Execute queries defined in the BPR security model, since the adversary does not need to decide from the beginning if all the exchanges will be passive or not. $\mathcal{A}$ can start with a passive sequence and decide at some point to stop (passive partial-session) or behave differently in an adaptive way (active session).

Resources of the Adversary. When doing security analyses, for every adversary and its privacy and authentication advantages, one also has to specify the adversarial resources such as the running time t, the number of oracle queries, the number of player instances, and the numbers $n_{\mathsf{passive}}/n_{\mathsf{active}}$ of (fully) passive and active sessions the adversary needs.

Discussion. We discuss a bit more about our security definitions to explain why they capture the practical threats. First, a passive network adversary is able to observe legitimate communications via SendServ and SendTerm queries (these will satisfy the passive sessions definition). An active network adversary can modify legitimate messages or impersonate a terminal or a server by injecting some messages of its choice, again, via SendServ and SendTerm queries. This models, in the standard way, possible insecurity (in terms of privacy or authentication) of the network channel between terminals and servers.

Passive-insider attacks (such as keylogger and screen capture malware compromising computers or their browsers) are modeled by Compromise queries followed by SendTerm queries. The former gives the adversary full information about the terminal's internal state, including its random coins and registers' contents, and the latter reveals to the adversary the inputs from the human.

We consider even more powerful attackers who can take full control of the computers or some of their crucial applications such as browsers. In this case, in addition to learning the internal state and all the inputs, the adversary can impersonate the honest terminal while sending adaptively selected messages to the human. We model this by Infect and SendHum queries.

Our model captures all the above scenarios and moreover, it takes into account the possibility of multiple simultaneous attacks, such as colluding network and malware adversaries. One can notice that attacks involving Infect queries are stronger than those with $\mathsf{Compromise}$ queries: when an adversary infects a terminal, it takes full control on it, with knowledge of its internal state, and thus plays on its behalf, using $\mathsf{SendServ}$ and $\mathsf{SendHum}$ queries.

Note that in any case, we are concerned with the security of a new session, in terms of privacy and authentication, over an honest terminal, that is neither compromised nor infected. Such security should be guaranteed even though the other sessions involving the same human with the same long-term secret were carried over compromised terminals, and if possible even over infected terminals. We model privacy via the Test query and with the appropriate privacy advantage definition. We model authentication via the corresponding advantage definition.

We also stress that we do not consider corruption of the long-term secrets, since they are known by the users and the server only, and we do not allow to corrupt them. Would the long-term secret be leaked, we cannot guarantee any security for future sessions. The interesting open problem of dealing with such corruptions could be addressed using an asymmetric long-term secret: a verifier-based variant that would just provide an encoded version of the user's secret to the server.

3.3 Human-Compatible Function Family

In this section, we define the syntax and security of a *Human-Compatible (HC)* function family and its instantiations.

3.3.1 Syntax

A HC function family is specified by the challenge space $\mathcal{C}$, the key generation algorithm $\mathcal{KG}$, which takes input the security parameter and outputs a key K, and the *challenge-response* function F that takes a key K and a challenge $x \in \mathcal{C}$ and returns the response $r = F_K(x)$. We require that:

1. for every K output by $\mathcal{KG}$ and every $x \in \mathcal{C}$, both x and $F_K(x)$ are human-writable and human-readable;
2. $\mathcal{C}$ is human-sampleable.

We also define the *only-human* HC function family (where an additional device is excluded), which is the human-compatible function family that also has:

1. for every K output by $\mathcal{KG}$, $F_K(\cdot)$ is human-computable;
2. every K output by $\mathcal{KG}$ is human-memorizable;

3.3.2 Security

In an authentication protocol with challenge-response pairs, intuitively, we would like that any successful authentication to a server should involve an evaluation of the function by the human user. So we expect no compromised/infected terminal to successfully authenticate to the server one more time than it interacted with the human. The security notion from the function is thus a kind of one-more unforgeability [77]. But here, any query to an $F_K(\cdot)$ oracle should help to immediately answer $F_K(x)$ to the current challenge x, since a second challenge will come from a new session that has closed the previous one, and so the previous challenge is obsolete: the adversary cannot store the $n + 1$ challenges, ask n queries, and answer the $n + 1$ initial challenges. In our protocols, the adversary gets a random challenge ($\mathsf{GetRandChal}$ query), can ask any $F_K(\cdot)$ query ($\mathsf{GetResp}$ query), but should answer that challenge ($\mathsf{TestResp}$ query), otherwise the failure is detected. After too many failures (recorded in the unvalidated-query counter ctr) one may restrict oracle queries. Hence our following security notion which formalizes these restrictions to the adversary.

η-Unforgeability. As said above, we thus define a kind of sequential one-more unforgeability experiment, with a limit η on the unvalidated-query counter ctr, where the queries follow the graph presented on Figure 3.1. Given a HC function family F, an adversary $\mathcal{A}$,

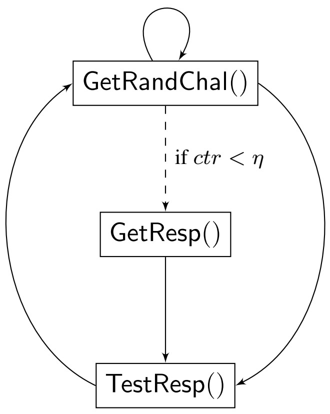


Figure 3.1: Graph of the sequential oracle calls in the η-unforgeability experiment

and a public parameter η, one first generates K with $\mathcal{KG}$ and initializes $ctr \leftarrow 0$. Then the adversary can ask the following queries, with possible short loops on the GetRandChal query and direct TestResp attempt right after getting the challenge:

1. GetRandChal() – It picks a new $x \overset{\$}{\leftarrow} \mathcal{C}$, marks it *fresh* and outputs it;
2. GetResp(x^*) – If $ctr < \eta$ and $x^* \in \mathcal{C}$, it returns $F_K(x^*)$ and increments ctr. It also marks the *fresh* x as *unfresh*. Otherwise, it outputs $\perp$;
3. TestResp(r) –
   - If $F_K(x) = r$ and x is *fresh*, the adversary *wins*;
   - If $F_K(x) = r$ and x is *unfresh*, it decrements ctr, marks x as *used*, and outputs 1;
   - Otherwise, it outputs 0.

Because of the sequential iterations, any TestResp query relates to the previous GetRandChal query. One can thus consider one memory-slot to store the challenges, but one only at a time: any new challenge replaces the previous one. The dashed line from GetRandChal to GetResp emphases the restriction on the number of unvalidated queries. When $ctr \geq \eta$, the adversary has no more choice than immediately trying an answer for the random challenges. The bound η represents the maximum gap that is allowed at any time between the number of GetResp queries and the number of correct TestResp queries. Note that a random challenge x can only be either *fresh*, *unfresh*, or *used*, and that marking it as one of those erases the other flags. Intuitively, a *fresh* challenge has not been compromised in any way, and succeeding at a TestResp on it would indicate the unforgeability has been breached, hence the winning status for the adversary, and the experiment stops. A challenge can switch to the *unfresh* state if the adversary asks the GetResp oracle for an answer. There are only two ways for the experiment to stop: if the adversary wins with a correct TestResp query on a fresh challenge; or if the adversary aborts, it then looses the game. We stress that the adversary can query the GetResp oracle on any x^* of its choice, and so possibly different from the current challenge x obtained with the previous GetRandChal query. But we give it a chance to still answer correctly to the challenge x with the correct TestResp query that, on an *unfresh* challenge, cancels the increment of the counter ctr. This counter represents the gap between the number of GetResp queries and the number of correct TestResp queries on random challenges. When one limits ctr to be at most 1, any GetResp query should be immediately followed by a correct TestResp query (one-more unforgeability).

This definition is a weaker notion than the one-more unforgeability [77], but still allows the adversary to exploit malleability: For example, with the RSA function, for a random challenge y, the adversary can ask a GetResp query on any $y' = y \cdot r^e \bmod n$, for an r of its choice, so that it can then extract an e th root of y. But this would not help it to answer a next fresh challenge.

2-Party η-Unforgeability. Unfortunately, the above clean security notion is not enough for our applications, as client-server situations and man-in-the-middle attacks allow more complex ordering of the queries by the adversary. We therefore present a variant of this experiment below, that is suitable for a protocol involving two parties (hence in the following $b \in \{0, 1\}$).

Given a HC function family F, an adversary $\mathcal{A}$, and a public parameter η, one first generates K with $\mathcal{KG}$ and initializes $ctr \leftarrow 0$. Then the adversary can ask the following queries:

1. $\mathsf{GetRandChal}(b)$ – It picks a new $x_b \xleftarrow{\$} \mathcal{C}$, marks it *fresh* and outputs it;

2. $\mathsf{GetResp}(x^*)$ – If $ctr < \eta$ and $x^* \in \mathcal{C}$, it returns $F_K(x^*)$ and increments ctr. It also marks all *fresh* x_b as *unfresh*. Otherwise, it outputs $\perp$;

3. $\mathsf{TestResp}(r, b)$ – If x_b exists:
   - If $F_K(x_b) = r$ and x_b is *fresh*, the adversary *wins*;
   - If $F_K(x_b) = r$ and x_b is *unfresh*, it decrements ctr, marks x_b as *used* and outputs 1;
   - Otherwise, it outputs 0.

The main difference with the previous experiment are the two memory-slots for challenges x_0 and x_1. But still, any GetResp query must be followed by a correct TestResp query to limit ctr from increasing too much.

The advantage of any adversary $\mathcal{A}$ against the unforgeability, $\mathbf{Adv}_F^{\eta\text{-}\mathsf{uf}}(\mathcal{A})$ is the probability of winning in the above experiment (with a correct TestResp query on a *fresh* challenge). Such a success indeed means that the adversary found the response for a new random challenge, without having asked for any GetResp query.

The resources of the adversary are the polynomial running time and the numbers $q_{\mathsf{c}}, q_{\mathsf{r}}, q_{\mathsf{t}}$ of queries to GetRandChal, GetResp and TestResp oracles, respectively.

Indistinguishability. For some constructions, we will expect the sequence of answers $\{F_K(x_i), i = 0, \ldots, T\}$ for challenges x_i (either adversarially chosen or not) to look random, or at least any new element in the sequence is not easy to predict from the previous ones.

For simplicity, we assume that there exists a global distribution $\mathcal{D}$ with large enough entropy D such that any such sequence is computationally indistinguishable from $\mathcal{D}^{T+1}$: We denote $\mathbf{Adv}_F^{\textsf{dist-}c}(\mathcal{D}, \mathcal{A})$ the advantage the adversary $\mathcal{A}$ can get in distinguishing the sequence $\{y_0 = F_K(x_0), \ldots, y_{c-1} = F_K(x_{c-1})\}$ for a random K, from $(y_0, \ldots, y_{c-1}) \overset{\$}{\leftarrow} \mathcal{D} \times \ldots \times \mathcal{D}$. For the latter distribution, the probability to guess y_{c-1} from the view of $(y_0, \ldots, y_{c-2})$ is $1/2^D$.

(Weak) pseudo-random functions definitely satisfy this property. But from a more practical point of view, the function implemented in the RSA SecurID device [68] is believed to satisfy it too, with x_i being a time-based counter.

3.3.3 Token-Based HC Function Family Instantiation

Then, we introduce a simple token-based HC function family. This assumes that the human is in possession of a simple device on which it can input challenge x and get the response $r \leftarrow F_K(x)$. The device will store K and perform the computation, but the human is still responsible for the communication with the terminal.

This allows us to use strong cryptographic primitives. For instance, we could set $K \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and $F_K : \{0, 1, \ldots, 9\}^{t'} \to \{0, 1, \ldots, 9\}^t$ a pseudorandom function. In the random oracle model (for modeling $\mathcal{H}$ in $F_K(x) = \mathcal{H}(K \| x)$), we have $\mathbf{Adv}_F^{\eta\textsf{-uf}}(\mathcal{A}) \leq 10^{-t}$ for any adversary and any η, since an adversary can just guess by chance the answer to a fresh challenge. Note that this function is obviously *human-readable*, *human-writable* and *human-sampleable* as its input/output are numbers in basis 10 so it is a HC function family.

Hence this function family is a good candidate to use in our Basic HAKE protocol (see Subsection 3.4.1) and its simplified version (see Subsection 3.5.1).

3.3.4 Only-Human HC Function Family Instantiation

However, avoiding such devices would be much better in practice. We are thus interested in the only-human HC function family that would not require anything beyond simple human memory and brain computation power. Since such a function is necessarily weaker, we will use it in our Confirmed HAKE protocol (see Subsection 3.4.2), which has a much tighter control over adaptive queries and therefore requires weaker security properties from the HC function family.

Construction

We present a candidate based on the construction of Blocki *et al.* [18], for which the security is based on [76]: Consider a challenge space $\mathcal{C} = \mathcal{X}_l^t \subseteq [n]^{lt}$, where $[n] = \{1, \ldots, n\}$ is the set of n integers each representing one of the n variables and $\mathcal{X}_l$ denotes the space of ordered clauses of l variables without repetition. The parameter t indicates that each challenge consists of t independent clauses, i.e., "small" challenges. The key generation algorithm $\mathcal{KG}$ of our HC function family takes as input a parameter n, then outputs a random mapping $\sigma : [n] \to \mathbb{Z}_d$ as the key K, where the integer d is a constant. Usually we set $d = 10$ because most humans are familiar with computations on digits. Let $\sigma^l : [n]^l \to \mathbb{Z}_d^l = (\sigma, \cdots, \sigma)$ denote the mapping that applies σ to each element of an l-tuple. Using a public human-computable function $f : \mathbb{Z}_d^l \to \mathbb{Z}_d$ that is instantiated later, the challenge-response function F takes a key $K = \sigma$ and a challenge $x \in \mathcal{C}$ as inputs, and returns a response $r = F_K(x)$. Here $F_K : \mathcal{C} \to \mathbb{Z}_d^t$ is defined as a t-tuple ($t \geq 1$) $(f \circ \sigma^l, \cdots, f \circ \sigma^l)$, where $\circ$ indicates the function composition.

For instance, if $n = 100$, $l = 3$, $d = 10$, $t = 2$, $x = ((1, 4, 20), (3, 36, 41))$, $\sigma(i) = (i + 3) \mod 10$ and $f = (x_1 - x_2 + x_3) \mod 10$, then $\sigma((1, 4, 20)) = (4, 7, 3)$, $\sigma((3, 36, 41)) = (6, 9, 4)$ and $F_K(x) = (0, 1)$.

Given integers $k_1, k_2 > 0$, the function f is instantiated as $f_{k_1,k_2} : \mathbb{Z}_{10}^{10+k_1+k_2} \to \mathbb{Z}_{10}$, which is defined as follows:

$$f_{k_1,k_2}(x_0, \ldots, x_{9+k_1+k_2}) = x_{\left(\sum_{i=10}^{9+k_1} x_i \mod 10\right)} + \sum_{i=10+k_1}^{9+k_1+k_2} x_i \mod 10.$$

Note that when f is instantiated as f_{k_1,k_2}, we have $l = 10 + k_1 + k_2$ and $d = 10$.

It is easy to see that such a function family is an only-human HC function family apart from the human memorization property. However, we can allow for images to represent the variables. As illustrated in [18], by using mnemonic helpers, humans are able to remember such mappings from images to digits. As an evidence, the primary author of [18] was able to memorize a mapping from $n = 100$ images to digits in 2 hours.

Security

In [18], the authors proved the intractability to answer to a new random challenge for the above HC function family instantiation based on the conjecture about the hardness of *random planted constraint satisfiability problems (RP-CSP)*. We briefly recall a special case of the RP-CSP conjecture, which we call the RP-CSP* conjecture, and its implied security theorem, both with our notations. For an in-depth review of those notions, the reader should refer to [18].

The RP-CSP* Conjecture. Before stating this conjecture, we introduce some notations as in [18]. Denote $H(\alpha_1, \alpha_2) = |\{i \in [n] \mid \alpha_1[i] \neq \alpha_2[i]\}|$ as the Hamming distance between two strings $\alpha_1, \alpha_2 \in \mathbb{Z}_d^n$. Use $H(\alpha) = H(\alpha, \vec{0})$ to denote the Hamming weight of α. Then we say two mappings $\sigma_1, \sigma_2 \in \mathbb{Z}_d^n$ are *ε-correlated* if $H(\sigma_1, \sigma_2)/n \leq (d-1)/d - \varepsilon$.

Conjecture 1 (RP-CSP*)**.** *Consider the function f_{k_1,k_2} described above, for any $\varepsilon, \varepsilon' > 0$ and any probabilistic polynomial time (in n) adversary $\mathcal{A}$, there exists an interger $N \in \mathbb{N}$, such that for all $n > N$, $m \leq n^{\min\{(k_2+1)/2, k_1+1-\varepsilon'\}}$, we have $\mathbf{Adv}^{\mathsf{rand}}_{f_{k_1,k_2}}(\mathcal{A}, \varepsilon) = \mathsf{negl}(n)$, where $\mathbf{Adv}^{\mathsf{rand}}_{f_{k_1,k_2}}(\mathcal{A}, \varepsilon)$ is the probability that $\mathcal{A}$ outputs a mapping σ' that is ε-correlated with the secret mapping σ given m random "small" challenge-response pairs $\{(C_i, f_{k_1,k_2}(\sigma^l(C_i)))\}_{1 \leq i \leq m}$.*

REMARK. The RP-CSP conjecture in [18] is a general version of the RP-CSP* Conjecture 1, where f can be instantiated as other functions. Here, for simplicity, we only state the conjecture where $f = f_{k_1,k_2}$. In [18], the authors also prove strong evidence in support of the RP-CSP conjecture: it holds for any statistical adversary and any Gaussian Elimination adversary. As observed in [76], most natural algorithmic techniques have statistical analogues except the Gaussian Elimination.

Basic η-Unforgeability. To state the security theorem in [18], we need the following "basic" HC security notion that is a "non-malleable" version of the η-unforgeability. It indeed assumes that asking a GetResp query with an input different from the current random challenge should not help to answer this challenge correctly to the TestResp query. Given a HC function family F, an adversary $\mathcal{A}$, and a public parameter η, one first generates K with $\mathcal{KG}$ and initializes $ctr \leftarrow 0$. Then the adversary can ask the following queries:

1. $\mathsf{GetRandChal}()$ – It picks a new $x \xleftarrow{\$} \mathcal{C}$, marks it *fresh* and outputs it;
2. $\mathsf{GetResp}(x^*)$ – It increments ctr if $x^* \neq x$;
   - If $ctr \leq \eta$ and $x^* \in \mathcal{C}$, it outputs $F_K(x^*)$ and marks x as *unfresh*;
   - Otherwise, it outputs $\perp$;
3. $\mathsf{TestResp}(r)$ –
   - If $F_K(x) = r$ and x is *fresh*, the adversary *wins*;
   - If $F_K(x) = r$ and x is *unfresh*, it outputs 1;
   - Otherwise, it outputs 0.

Just like the η-unforgeability experiment, the above oracle calls are sequential (similar to Figure 3.1), starting with a GetRandChal query. But since non-malleability is assumed, only GetResp queries with inputs different from the current random challenges make the counter increase, and it is never decreased.

The advantage of any adversary $\mathcal{A}$ against the above unforgeability, $\mathbf{Adv}_F^{\eta\text{-uf-basic}}(\mathcal{A})$ is the probability of winning in the above experiment. Such a success indeed means that the adversary found the response for a new random challenge, without having asked for a $\mathsf{GetResp}$ query. The parameter η restricts the number of "adaptive" $\mathsf{GetResp}$ queries that $\mathcal{A}$ can make, where adaptive means "different from the current random challenge".

The resources of the adversary are the polynomial running time and the numbers $q'_\mathsf{c}, q'_\mathsf{r}, q'_\mathsf{t}$ of queries to the above $\mathsf{GetRandChal}$, $\mathsf{GetResp}$ and $\mathsf{TestResp}$ oracles, respectively. For convenience, denote by q''_t the number of $\mathsf{TestResp}$ queries such that the current random challenge x is *fresh*. By definition, we have $q'_\mathsf{r} \le q'_\mathsf{c}$, $q'_\mathsf{t} \le q'_\mathsf{c}$ and $q''_\mathsf{t} \le q'_\mathsf{c} - q'_\mathsf{r}$.

HC Function Family Security Results. Under Conjecture 1, one can prove the following unforgeability result about the HC function family.

Theorem 1 (From [18]). *Given $\varepsilon, \varepsilon' > 0, t \in \mathbb{N}_+$ and $\delta > (\frac{1}{10} + \varepsilon)^t$, for any probabilistic polynomial time (in $n, q'_\mathsf{c}, 1/\varepsilon$) adversary $\mathcal{A}$ against the basic 0-unforgeability security of the HC function family F constructed above using $f = f_{k_1,k_2}$ with*

$$q''_\mathsf{t} = 1, q'_\mathsf{c} \le \frac{1}{t} \cdot n^{\min\{(k_2+1)/2, k_1+1-\varepsilon'\}} - 1,$$

under Conjecture 1, we have $\mathbf{Adv}_F^{\text{0-uf-basic}}(\mathcal{A}) < \delta$.

Note that in the basic 0-unforgeability security game, the adversary learns nothing from $\mathsf{GetResp}(x^*)$ if x^* is not the current random challenge x. So if $\eta = 0$, the adversary $\mathcal{A}$ is only given random challenge-response pairs.

This result is actually not strictly good-enough, even for our Confirmed HAKE protocol (see Subsection 3.4.2). Indeed, if the function does not allow for at least one adaptive query, an attacker could make it in the first exchange (using an infected terminal), then break the unforgeability of the function before the confirmation flow and make the protocol succeed, hence avoiding detection. Thus, we extend the RP-CSP conjecture to allow $\log n$ adaptive "small" challenge-response pairs.

Conjecture 2. *Consider the function f_{k_1,k_2} described above, for any $\varepsilon, \varepsilon' > 0, t \in \mathbb{N}_+$ and any probabilistic polynomial time (in n) adversary $\mathcal{A}$, there exists an interger $N \in \mathbb{N}$,* *such that for all $n > N$, $m_r \leq n^{\min\{(k_2+1)/2, k_1+1-\varepsilon'\}}$ and $m_a \leq t\log n$, we have $\mathbf{Adv}^{\mathsf{adapt}}_{f_{k_1,k_2}}(\mathcal{A}, \varepsilon) = \mathsf{negl}(n)$, where $\mathbf{Adv}^{\mathsf{adapt}}_{f_{k_1,k_2}}(\mathcal{A}, \varepsilon)$ is the probability that $\mathcal{A}$ outputs a mapping σ' that is ε-correlated with the secret mapping σ given m_r random "small" challenge-response pairs and the correct responses to m_a "small" challenges adaptively chosen by $\mathcal{A}$.*

Proof. For any adversary $\mathcal{A}$ we can construct an adversary $\mathcal{B}$ such that $\mathbf{Adv}^{\mathsf{adapt}}_{f_{k_1,k_2}}(\mathcal{A}, \varepsilon) \leq 10^{t\log n} \times \mathbf{Adv}^{\mathsf{rand}}_{f_{k_1,k_2}}(\mathcal{B}, \varepsilon)$.

$\mathcal{B}$ simulates $\mathcal{A}$'s view by providing $\mathcal{A}$ with the given m_r random "small" challenge-response pairs and randomly guessing the responses to the m_a ($\leq t\log n$) adaptive "small" challenges. The probability of correctly guessing all adaptive ones is $10^{-t\log n}$ (refer to the construction of f_{k_1,k_2}), hence the above advantage reduction. One should note that $10^{t\log n} \times \mathsf{negl}(n) = \mathsf{negl}(n)$ and $\mathcal{B}$'s running time is polynomial in n. □

Under this extended conjecture, one can prove the following stronger unforgeability result about the HC function family, which "almost" suits our Confirmed HAKE protocol (see Subsection 3.4.2):

Theorem 2. *Given $\varepsilon, \varepsilon' > 0, t \in \mathbb{N}_+$ and $\delta > (\frac{1}{10} + \varepsilon)^t$, for any probabilistic polynomial time (in $n, q'_{\mathsf{c}}, 1/\varepsilon$) adversary $\mathcal{A}$ against the basic η-unforgeability security of the HC function family F constructed above using $f = f_{k_1,k_2}$ with*

$$\eta \leq \log n, q'_{\mathsf{c}} \leq \frac{1}{t} \cdot n^{\min\{(k_2+1)/2, k_1+1-\varepsilon'\}} - 1,$$

under Conjecture 2, we have $\mathbf{Adv}^{\eta\text{-uf-basic}}_{F}(\mathcal{A}) < q''_{\mathsf{t}} \cdot \delta$.

Proof. The proof is almost the same as that of Theorem 1. Informally, we need to show that any adversary $\mathcal{A}$ that breaks the basic η-unforgeability security of the HC function family can also "recover" the secret mapping σ in Conjecture 2. The reader can refer to the proof of Theorem 5 in [18], which we call the "HCP" proof below, for the details.

Nevertheless, here the theorem differs from Theorem 1 in several aspects. First, $\mathcal{A}$ can adaptively select $t \log n$ "small" challenges to get the correct responses, while in Theorem 1 only random ones are allowed. But having adaptive queries does not affect the HCP proof because it only uses $\mathcal{A}$ as a blackbox to predict the responses to any t "small" challenges. Second, we apply an union bound of q''_{t} queries to the final advantage. $\square$

REMARK. In the above theorem $\varepsilon, \varepsilon'$ are almost 0. We can set $n = 100$, $k_1 = 1$, $k_2 = 3$ and $t = 5$, then $\eta \leq 6$, $q'_{\mathsf{c}} \leq n^2/t - 1 \approx 2000$ and $\mathbf{Adv}_F^{\eta\text{-uf-basic}}(\mathcal{A}) < q''_{\mathsf{t}} \cdot 10^{-t} \leq 1/50$.

We believe a similar theorem holds for $\mathbf{Adv}_F^{\eta\text{-uf}}$ (by replacing the oracles with those in the 2-party η-unforgeability experiment), which our HAKE security can rely on. The intuition is as follows. With the HC function family instantiation described in this section, a $\mathsf{GetResp}(x^*)$ query in the η-unforgeability experiment should not have x^* too "far" from the random challenge x output by the latest $\mathsf{GetRandChal}$ query. Otherwise, it is very unlikely for the adversary to guess correctly in the $\mathsf{TestResp}$ query. But the adversary can modify x a little bit to guess the correct response with a smaller failure probability. This is the difference between the two unforgeability notions: the basic one does not tolerate any malleability, whereas the other can exploit malleability. Because of the size of the challenge space, that has to be quite large (it is essentially $n^{t(10+k_1+k_2)}$, and thus 2^{465}, with the above parameters), the number of challenges that are "close" to any random challenge accounts for a tiny proportion. Thus, the adversary should not get much help from such "nearly random" challenges. Besides, such queries risk increasing the counter in the $\mathsf{GetResp}$ oracle without extracting much useful information. In addition, the two memory slots will not increase much the advantage of an adversary, and so $\mathbf{Adv}_F^{\eta\text{-uf-basic}}$ and $\mathbf{Adv}_F^{\eta\text{-uf}}$ should be quite close for this specific HC function family instantiation. We leave further studies of security of the HC function family from [18] to future works.

3.4 Generic HAKE Protocols

In this section, we propose two generic HAKE protocols. They build on a simple idea of composing a human-compatible (HC) function family with a password authenticated key exchange (PAKE) protocol. More precisely, a server chooses a random challenge x, the user U_ℓ's response is $r = F_{K_\ell}(x)$, where F is a HC function family and K_ℓ is the long-term secret shared between the user and the server. And finally the terminal and the server execute the PAKE on the one-time password r, as in [75]. As already mentioned, whereas the server supports concurrent sessions, since the human does not, there is no sense in maintaining multiple session states for one human user.

However, a straightforward replay attack is possible. The adversary can first just eavesdrop a session by compromising a terminal, and then play on behalf of the server with the observed challenge-response pair (x, r), even when the user uses an honest terminal. The main issue is that there is no reason for the challenge to be distinct in the various sessions if we do not add a mechanism to enforce it. In [75]'s constructions, they assume the server is stateful to prevent it. However, we can do better.

This is the goal of our first protocol: it adds a coin-flipping protocol between the terminal and the server to avoid either party to influence the challenge x, and thus to avoid the aforementioned replay attacks. We prove it secure (in terms of privacy, which implies implicit authentication) assuming security of commitments (underlying the coin-flipping), HC function family, and PAKE. However, the concrete security depends on the bound η, which is large enough for our device-based HC function family, but the only-human HC function family construction we proposed in Subsection 3.3.4 (and its underlying hardness problem) does not tolerate a high η.

Hence, the goal of our second HAKE protocol is to add explicit authentication, which will help limiting the number of malicious challenge-response pairs the adversary can see, or at least to detect them: the user can then suspect the terminal to be infected. We still

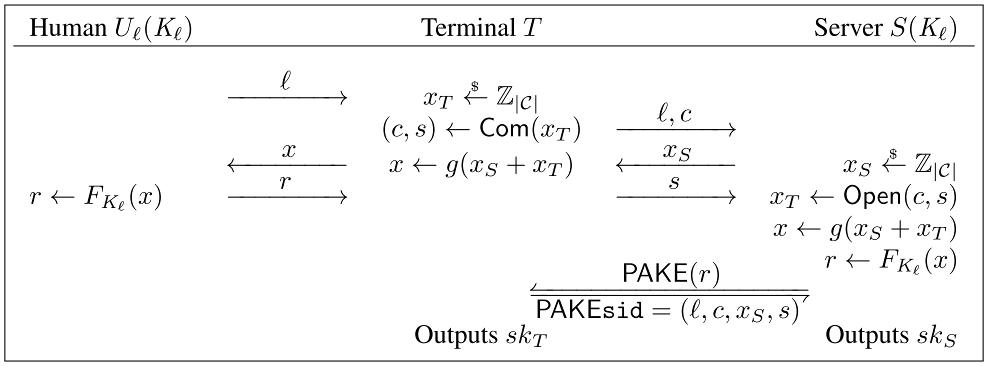

Human $U_\ell(K_\ell)$
Terminal T
Server $S(K_\ell)$
ℓ
$x_T \xleftarrow{\$} \mathbb{Z}_{|\mathcal{C}|}$
$(c,s) \leftarrow \mathsf{Com}(x_T)$
ℓ, c
x
$x \leftarrow g(x_S + x_T)$
x_S
$x_S \xleftarrow{\$} \mathbb{Z}_{|\mathcal{C}|}$
r
s
$r \leftarrow F_{K_\ell}(x)$
$x_T \leftarrow \mathsf{Open}(c,s)$
$x \leftarrow g(x_S + x_T)$
$r \leftarrow F_{K_\ell}(x)$
$\mathsf{PAKE}(r)$
$\mathsf{PAKEsid} = (\ell, c, x_S, s)$
Outputs sk_T
Outputs sk_S

Figure 3.2: Basic HAKE construction

need the concrete HC function to tolerate at least one malicious challenge, but this remains a reasonable assumption.

3.4.1 The Basic HAKE

Our first construction utilizes a commitment scheme, a HC function family, and PAKE.

Description. Let $(\mathcal{KG}, F)$ be a human-compatible function family with challenge space $\mathcal{C}$, let $\mathcal{CS} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open})$ be a commitment scheme, let PAKE be a password authenticated key exchange protocol and let $g : \mathbb{Z}_{|\mathcal{C}|} \to \mathcal{C}$ be a bijection. We construct the Basic HAKE $(\mathcal{LKG} = \mathcal{KG}, \mathcal{KE})$. Its interactive $\mathcal{KE}$ protocol is described on Figure 3.2, here are the descriptions.

- $\mathcal{KE}$ execution:

  1. When the user invokes a terminal to establish a connection with the server, the terminal chooses its part of the challenge x_T, and commits it for the server. It also sends the user's identifier ℓ;

  2. Upon receiving the commitment, the server waits until any previous session for U_ℓ finishes, then it chooses its part of the challenge x_S, and sends it in clear to the terminal;

  3. The terminal then combines both parts x_T and x_S to generate the challenge $x = g(x_S + x_T)$, and asks x to the user;

4. Upon reading the challenge x, the user computes and writes down the response r for the terminal;

5. When the terminal receives the response r from the human user, it opens its commitment to the server, and can already starts with the PAKE protocol execution;

6. Upon receiving the opening value of the commitment, the server opens the latter to get x_T. It can then combine both parts x_T and x_S to generate the challenge $x = g(x_S + x_T)$, and compute the response r. It can then proceed with the PAKE protocol too.

The terminal and the server both run the PAKE protocol with their (expected) common input r and session identifier $\mathsf{PAKE}\texttt{sid}$ that is the concatenation of the transcript. At the end of the PAKE execution, they come up with two session keys, sk_T and sk_S, respectively, that will be equal if both parties used the same r in the PAKE. Since we do not consider explicit authentication, accept and $\mathsf{terminate}$ flags are not set.

Correctness of the HAKE construction follows from correctness of the building blocks.

Security Analysis. For Basic HAKE, we only assess privacy of the session key, since this protocol does not provide explicit authentication. We have the following theorem and refer to [39] for the proof.

Theorem 3. *Consider the Basic HAKE protocol defined in Figure 3.2. Let $\mathcal{A}$ be an adversary against the privacy security game with static compromises, running within a time bound t and using less than n_{comp}* compromised *terminal sessions, n_{uncomp}* uncompromised *terminal sessions, n_{serv} server sessions and $n_{\mathsf{active}} \leq n_{\mathsf{comp}} + n_{\mathsf{uncomp}} + n_{\mathsf{serv}}$ active sessions. Then there exist an adversary $\mathcal{B}_1$ attacking the 2-party n_{comp}-unforgeability of the HC function family with $q_{\mathsf{r}}, q_{\mathsf{c}}, q_{\mathsf{t}}$ queries of the corresponding type, an adversary $\mathcal{B}_2$ and a distinguisher $\mathcal{B}_3$ attacking UC-security of* PAKE *with a simulator $\mathcal{S}_{\mathsf{PAKE}}$ as well as* *an adversary $\mathcal{B}_4$ against the commitment scheme, all running in time t, such that*

$$\mathbf{Adv}^{\mathsf{priv}}_{\mathsf{HAKE}}(\mathcal{A}) \leq \mathbf{Adv}^{n_{\mathsf{comp}}\text{-}\mathsf{uf}}_{F}(\mathcal{B}_1) + 2 \cdot \mathbf{Adv}^{\mathsf{pake}}_{\mathsf{PAKE}}(\mathcal{S}_{\mathsf{PAKE}}, \mathcal{B}_2, \mathcal{B}_3) + 6 \cdot \mathbf{Adv}_{\mathcal{CS}}(\mathcal{B}_4) \ ,$$

where $q_{\mathsf{r}} \leq n_{\mathsf{comp}}$, $q_{\mathsf{t}} \leq n_{\mathsf{active}}$, and $q_{\mathsf{c}} \leq n_{\mathsf{uncomp}} + n_{\mathsf{comp}} + n_{\mathsf{serv}}$.

Discussion. Concrete security of the HC function family is definitely the most crucial compared to that of other building blocks, since it is hard to balance strong security and usability. This is why we emphasize this in the above theorem.

We note that the sessions which lead to TestResp queries have *non-oracle-generated* flows and therefore correspond to classical on-line dictionary attacks: the adversary simply tries to impersonate the user/terminal to the server (or vice-versa), with a guess for the answer r (unless the query has been asked to GetResp). Indeed, sessions with GetResp queries on the exact challenges have *compromised* terminals and correspond to a spyware keylogger that records random challenge-response pairs. If using such a compromised terminal can be considered rare, this remains reasonable. Eventually, the sessions which lead to GetResp queries on different challenges are the most critical, but they should likely fail. And we expect them to be quite exceptional. As remarked above, such sessions likely conclude to a failure: no concrete session is established with the server the user wanted to connect to (if the HC function family is still secure after such adaptive queries). If the user can detect such a failure, he can run away from this terminal. We now propose a way for the user to detect such a dangerous terminal, and thereafter take appropriate measures. At the same time, our next proposal will achieve *explicit* authentication.

3.4.2 The Confirmed HAKE

We now enhance the Basic HAKE by adding two confirmation flows (see Figure 3.3) that allow the user to detect a bad behavior of the adversary, who compromised the device, and take appropriate measures. This can happen in two different scenarios: the adversary just

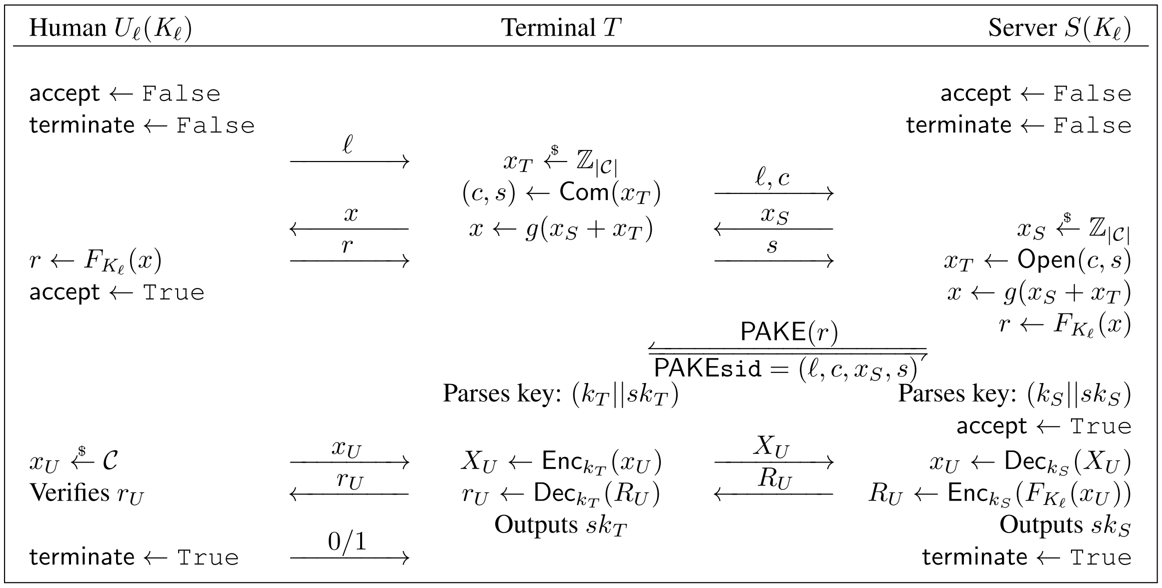

Human $U_\ell(K_\ell)$
Terminal T
Server $S(K_\ell)$
accept $\leftarrow$ False
terminate $\leftarrow$ False
accept $\leftarrow$ False
terminate $\leftarrow$ False
ℓ
$x_T \xleftarrow{\$} \mathbb{Z}_{|\mathcal{C}|}$
$(c, s) \leftarrow \mathsf{Com}(x_T)$
ℓ, c
x
$x \leftarrow g(x_S + x_T)$
x_S
$x_S \xleftarrow{\$} \mathbb{Z}_{|\mathcal{C}|}$
r
s
$r \leftarrow F_{K_\ell}(x)$
$x_T \leftarrow \mathsf{Open}(c, s)$
accept $\leftarrow$ True
$x \leftarrow g(x_S + x_T)$
$r \leftarrow F_{K_\ell}(x)$
$\mathsf{PAKE}(r)$
$\mathsf{PAKEsid} = (\ell, c, x_S, s)$
Parses key: $(k_T||sk_T)$
Parses key: $(k_S||sk_S)$
accept $\leftarrow$ True
$x_U \xleftarrow{\$} \mathcal{C}$
x_U
$X_U \leftarrow \mathsf{Enc}_{k_T}(x_U)$
X_U
$x_U \leftarrow \mathsf{Dec}_{k_S}(X_U)$
Verifies r_U
r_U
$r_U \leftarrow \mathsf{Dec}_{k_T}(R_U)$
R_U
$R_U \leftarrow \mathsf{Enc}_{k_S}(F_{K_\ell}(x_U))$
Outputs sk_T
Outputs sk_S
terminate $\leftarrow$ True
$0/1$
terminate $\leftarrow$ True

Figure 3.3: Confirmed HAKE construction

compromised a terminal and additionally plays on behalf of the server, which allows it to ask any query to the user through the terminal, or the adversary infected a terminal that allows it to directly ask any query to the user.

As said above, such dangerous cases lead to no connection with the expected server. The user will thus check whether he built a secure session with the expected server, who should be able to answer to a fresh random challenge. This is performed under the fresh key, established with the PAKE, using a secure authenticated encryption. As shown below, the two additional flows will not only provide *explicit* authentication, but also allow the user to detect such bad events and take measures. For this, it is important that the user does not start multiple sessions concurrently, which is anyway not realistic for a human (as already noticed above).

Description. The protocol is similar to Basic HAKE, but it uses an additional building block, an authenticated encryption scheme $\mathcal{ES} = (\mathsf{Enc}, \mathsf{Dec})$, that is used in the new last stage of the protocol. The description is in Figure 3.3.

Since we now consider the authentication of the players, we additionally include accept and terminate flags in the protocol: The user U_ℓ accepts after sending the first response while the server S accepts after the PAKE. Then both terminate when they have the confirmation of the other partner. More precisely, the user terminates after sending the last bit (1 for acceptance and 0 for rejection) to the terminal (thus having verified the server's response in the last stage), and the server terminates after sending the encrypted response (thus having checked the terminal can generate a valid ciphertext).

Note that if the protocol terminates, sk_T and sk_S must be equal, since our additional flows act as confirmation flows for the PAKE.

Security Analysis. We now present Theorem 4 regarding the security of our Confirmed HAKE in the HAKE privacy and authenticity experiment.

While it relies on the same security properties of PAKE, authenticated encryption, commitment scheme and HC function family, a critical parameter is added, the number of human sessions that *reject* in the end.

Indeed, the explicit authentication property we achieve means that any attempt at issuing an adaptive query unrelated to the challenge will likely lead to a failure of the PAKE protocol, that can in turn be detected by the human, as he doesn't get the answer to x_U he looked for. This allows to use a much stricted η in the HC unforgeability game (even $\eta = 1$ for a very strict human user), which is a much more reasonable goal for an only-human HC function family.

We have the following theorem and refer to [39] for the proof.

Theorem 4. *Consider the Confirmed HAKE protocol defined in Figure 3.3. Let $\mathcal{A}, \mathcal{A}'$ be adversaries against the privacy and authenticity security game of HAKE within a time bound t and using less than n_{comp}* compromised *terminal sessions, n_{uncomp}* uncompromised *terminal sessions, n_{serv} server sessions, $n_{\mathsf{active}} \leq n_{\mathsf{comp}} + n_{\mathsf{uncomp}} + n_{\mathsf{serv}}$ active sessions and n_{hr} human session that* reject *in the end. Then there exist two adversaries $\mathcal{B}_1, \mathcal{B}_1'$ attacking the 2-party $(n_{\mathsf{hr}} + 1)$-unforgeability of HC function family with $q_{\mathsf{r}}, q_{\mathsf{c}}, q_{\mathsf{t}}$ queries of the corresponding type, two adversaries $\mathcal{B}_2, \mathcal{B}_2'$ and two distinguishers $\mathcal{B}_3, \mathcal{B}_3'$ attacking UC-security of* PAKE *with the simulator $\mathcal{S}_{\mathsf{PAKE}}$, two adversaries $\mathcal{B}_4, \mathcal{B}_4'$ against the authen-*

ticated encryption, as well as two adversaries $\mathcal{B}_5, \mathcal{B}'_5$ against the commitment scheme, all running in time t, such that

$$\begin{aligned}\mathbf{Adv}^{\mathsf{priv}}_{\mathsf{HAKE}}(\mathcal{A}) \leq \mathbf{Adv}^{(n_{\mathsf{hr}}+1)\text{-}\mathsf{uf}}_{F}(\mathcal{B}_1) + 2(n_{\mathsf{comp}} + n_{\mathsf{uncomp}} + n_{\mathsf{serv}}) \cdot \mathbf{Adv}^{\mathsf{authenc}}_{\mathcal{ES}}(\mathcal{B}_4) \\ + 6 \cdot \mathbf{Adv}_{\mathcal{CS}}(\mathcal{B}_5) + 2 \cdot \mathbf{Adv}^{\mathsf{pake}}_{\mathsf{PAKE}}(\mathcal{S}_{\mathsf{PAKE}}, \mathcal{B}_2, \mathcal{B}_3), \\ \mathbf{Adv}^{\mathsf{auth}}_{\mathsf{HAKE}}(\mathcal{A}') \leq \mathbf{Adv}^{(n_{\mathsf{hr}}+1)\text{-}\mathsf{uf}}_{F}(\mathcal{B}'_1) + 2(n_{\mathsf{comp}} + n_{\mathsf{uncomp}} + n_{\mathsf{serv}}) \cdot \mathbf{Adv}^{\mathsf{authenc}}_{\mathcal{ES}}(\mathcal{B}'_4) \\ + 6 \cdot \mathbf{Adv}_{\mathcal{CS}}(\mathcal{B}'_5) + 2 \cdot \mathbf{Adv}^{\mathsf{pake}}_{\mathsf{PAKE}}(\mathcal{S}_{\mathsf{PAKE}}, \mathcal{B}'_2, \mathcal{B}'_3),\end{aligned}$$

where $q_{\mathsf{r}} \leq 2n_{\mathsf{comp}}$, $q_{\mathsf{t}} \leq n_{\mathsf{active}}$, $q_{\mathsf{c}} \leq n_{\mathsf{comp}} + n_{\mathsf{uncomp}} + n_{\mathsf{serv}}$.

REMARK. We also note that given the confirmation phase, and assuming the strong policy of resetting all credentials if the confirmation phase fails, the coin-flipping part is no longer necessary for the security proof: we could let the server choose the challenge during the first phase and the human in the second one (to avoid one player being to make replay attacks). We chose to keep it as part of the protocol because, first, this would not reduce the number of flows since the terminal always initiates such a connection, and second, without coin-flipping a network attacker could test adaptive challenges. The confirmation phase would fail, but there is no real need for the user to take severe measures and change the long-term secret in such a weak attack. Hence we prevent adaptive tests (from network attacks) with coin-flipping, which may be useful if a policy a little weaker is in use, such as resetting only if there is suspicion of terminal infection.

3.5 Device-Assisted HAKE Protocols

In this section, we take a step back from the only-human HC function family to allow the use of an additional device that will perform the computations in place of the human. In this setting, the HC function family can be quite powerful and thus resist to many adaptive queries. We consider it in two scenarios: first in a similar context as the Basic HAKE, where one can enter a challenge onto the device to get the response; and second, a time-based token, that outputs the response every timeframe, with the time as the challenge (without having the user to enter it).

3.5.1 Simplified Basic HAKE

According to the security proof of the Basic HAKE, the PAKE has to be instantiated with a UC-secure protocol, which turns out to be quite costly. Indeed, the only efficient scheme that achieves this security level is the encrypted key exchange protocol (EKE) [17]. However, the proof holds in the ideal cipher model, for a symmetric blockcipher that should only output elements in the Diffie-Hellman group. In practice, the best way to do it is to iterate a large blockcipher until one falls in the group. First, a large blockcipher from a hash function (modeled as a random oracle) has fueled a whole line of works [78, 79, 80], and is nevertheless already quite costly: at the time of writing, at least 8-round Feistel network is required [80], with an impossibility result below 6 [78]. Thereafter, additional iterations are required to build a permutation onto the group. This thus eventually corresponds to dozens of hash function evaluations.

Looking back at the construction, using a full PAKE seems anyway as a bit of an overkill since the ephemeral secrets are only used once, and need not to be kept secret afterwards. We hence propose a simplified Basic HAKE protocol that uses commitments instead of a full PAKE to achieve better efficiency. Due to its very similarity to the Basic HAKE and that "simplified PAKE" is also used in our Time-Based HAKE described below, for conciseness we refer to [39] for the details.

3.5.2 Time-Based HAKE

Scenario. In this section, we focus on the particular (but quite usual) case where the physical device does not have a dedicated input but uses time instead to compute its output. More precisely, our protocol considers a device, such as the RSA-SecurId [68] token, that, based on an internal seed (the long-term key K_ℓ), generates a one-time password (the value

Time	Human U_ℓ		Terminal T		Server S
$\leq t$	$\mathsf{accept} \leftarrow \texttt{False}$				$\mathsf{accept} \leftarrow \texttt{False}$
					$\mathsf{terminate} \leftarrow \texttt{False}$
t		$\xrightarrow{\ell}$	$x_T \xleftarrow{\$} \mathbb{Z}_p, X_T \leftarrow g^{x_T}$		$x_S \xleftarrow{\$} \mathbb{Z}_p, X_S \leftarrow g^{x_S}$
t		$\xrightarrow{pw_t}$	$(c_T, s_T) \leftarrow \mathsf{Com}_T(X_T, pw_t)$	$\xrightarrow{\ell, X_T, c_T}$	
t	$\mathsf{accept} \leftarrow \texttt{True}$			$\xleftarrow{X_S, c_S}$	$(c_S, s_S) \leftarrow \mathsf{Com}_S(X_S, pw_t)$
$\vdots$			Wait for timeframe $> t$		Wait for timeframe $> t$
$> t$				$\xrightarrow{s_T}$	If $\mathsf{Open}_T(c_T, s_T) = (X_T, pw_t)$
$> t$					and $(\ell, t) \notin \Lambda$, store (ℓ, t) in Λ
$> t$					Otherwise reject
$> t$					$\mathsf{accept} \leftarrow \texttt{True}$
$> t$			Reject if $\mathsf{Open}_S(c_S, s_S)$	$\xleftarrow{s_S}$	Outputs $(X_T)^{x_S}$
$> t$			$\neq (X_S, pw_t)$		
$> t$			Outputs $(X_S)^{x_T}$		$\mathsf{terminate} \leftarrow \texttt{True}$

Figure 3.4: Time-Based HAKE construction

$F_{K_\ell}(t)$, based on the time period t), and displays it on an LCD-Screen. The password is tied to an internal clock, and changes every τ (e.g. 30s). Note that such a password is already *human readable* and *human writable*, hence it satisfies our human-compatible communications.

Building on the security model presented in Section 3.2, we now consider time as a variable, that is to be segmented into timeframes (each spanning τ seconds). We then number those timeframes and associate to each message sent between T and S this number, representing the fact that each party can measure time and identify the timeframe in which the message was sent.

Since the one-time passwords are generated by a secure device implementing F_{K_ℓ}, we can make the assumption that, for each timeframe, the output is indistinguishable from an element sampled from the distribution $\mathcal{D}$ with entropy greater than D (which increases the advantage of an adversary $\mathcal{A}$ by at most $\mathbf{Adv}_F^{\mathsf{dist}\text{-}T}(\mathcal{A})$ after T timeframes).

We rely on the requirement that any user U_ℓ can only make use of one terminal during a timeframe. That is, he may not attempt to authenticate using more than one terminal in a single time period.

Protocol. We now propose a device-assisted HAKE protocol called Time-Based HAKE. It is presented on Figure 3.4. As in the previous Simplified Basic HAKE, it makes use of a commitment scheme on top of the unauthenticated Diffie-Hellman scheme to perform authentication.

The commitment scheme $\mathcal{CS}$ is initialized twice, with two independent setup, leading to $\mathsf{Com}_T/\mathsf{Open}_T$ and $\mathsf{Com}_S/\mathsf{Open}_S$, each of them being used for the commitments generated by the terminal and the server, respectively. We also setup a group $\mathbb{G}$ of prime order p in which the discrete logarithm problem is believed to be hard. Let g be a generator of $\mathbb{G}$.

The protocol itself is split into two parts: the *commitment* phase which must happen during a timeframe t (that we will call the *session timeframe*) and the *verification* phase, that must happen later than the session timeframe.

This delay is a clear limitation on the total speed of the protocol, which on average will take $\tau/2$. It will however prove necessary, as it allows $F_{K_\ell}(t)$ to be revealed without compromising the security of the scheme, therefore building on the one-time specificity of the password. To enforce a unique session in a timeframe, the server will not accept to run several sessions within the same timeframe, with the same user, as the latter should not do it anyway (see above). This would thus come from an adversary, and then allowing multiple sessions in a timeframe t can compromise other sessions in the same timeframe when $\mathcal{F}_{K_\ell}(t)$ is revealed.

It is interesting to note that this protocol uses the time period t as the HAKE challenge (the challenge is a counter) and the one-time password ($F_{K_\ell}(t)$) read from the device as the human's response. Therefore, partnering between U and S is entirely determined at the end of the session timeframe t.

Security Analysis. In the security analysis, as in Section 3.2.1, we only consider static *compromises*. Hence $\mathsf{Compromise}(j)$ can only be the first oracle query of a session, and $\mathsf{Infect}(j)$ can only affect *compromised* sessions. Since *compromises* are known before the first flow and partnering between Human and Server is determined at the end of timeframe t, this means that *freshness* itself can be perfectly ascertained in any timeframe $> t$.

We have the following theorem and refer to [39] for the proof.

Theorem 5. *Consider the Time-Based Device-Assisted HAKE protocol defined in Fig-*

Table 3.1: Performance of the Time-Based Device-Assisted HAKE

Scheme	Flows	Terminal		Server		Communication complexity
		expon.	$\mathcal{H}$ eval.	expon.	$\mathcal{H}$ eval.	
`1(SPAKE1)` [75, 81]	4	3	1	3	1	4λ
This work	4	2	2	2	2	10λ

ure 3.4. Let $\mathcal{A}, \mathcal{A}'$ be an adversaries against the privacy and user authenticity security games with static compromises, running within time t and using less than n_{serv} non-passive sessions against the server oracle, n_{term} non-passive sessions against the terminal oracle, $n_{\mathsf{total}} > n_{\mathsf{term}} + n_{\mathsf{serv}}$ total sessions and $T < n_{\mathsf{total}}$ unique timeframes. Then there exist an adversary $\mathcal{B}_1$ against the indistinguishability of the password-distribution $\mathcal{D}$ running in time t, an adversary $\mathcal{B}_2$ against the commitment scheme running in time t, and an adversary $\mathcal{B}_3$ against the DDH experiment running in time $t + 8n_{\mathsf{total}}\tau_{exp}$:

$$\mathbf{Adv}^{\mathsf{priv}}_{\mathsf{HAKE}}(\mathcal{A}) \le (n_{\mathsf{serv}} + n_{\mathsf{term}}) \cdot 2^{-D} + \mathbf{Adv}^{\mathsf{dist}\text{-}T}_{F}(\mathcal{B}_1) + 4 \cdot \mathbf{Adv}_{\mathcal{CS}}(\mathcal{B}_2) + \mathbf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{B}_3),$$
$$\mathbf{Adv}^{\mathsf{u\text{-}auth}}_{\mathsf{HAKE}}(\mathcal{A}') \le (n_{\mathsf{serv}} + n_{\mathsf{term}}) \cdot 2^{-D} + \mathbf{Adv}^{\mathsf{dist}\text{-}T}_{F}(\mathcal{B}_1) + 3 \cdot \mathbf{Adv}_{\mathcal{CS}}(\mathcal{B}_2),$$

with τ_{exp} the time necessary to exponentiate one group element, and n_{total} the global number of sessions.

REMARK. Note that the Time-Based Device-Assisted HAKE only achieves user authentication in our setting, since *server authentication* requires the server identity to be approved by the human in our setting (the terminal could be *infected* so it cannot be relied on). A similar approach to the one of the Confirmed HAKE could be used to achieve a full mutual authentication.

Perfomances. We offer in Table 3.1 a comparison (in terms of numbers of flows, exponentiations, $\mathcal{H}$ evaluations and overall communication complexity) of the performances of our HAKE protocol with the one-time PAKE `1(P)` construction of [75], instantiated with `SPAKE1` from [81] as a reference.

Since `SPAKE1` is also proven in the random oracle model, it is fair to use the efficient commitment scheme described in Section 2.3. We do not include the redundant X_P in s_P (it is transmitted at the commitment stage) for the communication complexity, and for a security parameter λ, we assume the group elements to be encoded into 2λ-long bit-strings.

While our communication complexity is higher, the computational load is reduced by 30% from [75] with the most efficient PAKE. Relaxing the PAKE security properties allows a significant gain from the complexity point of view.

CHAPTER 4
COMPARING TLS 1.3 OVER TFO TO QUIC

4.1 Introduction

As motivated in Subsection 1.2.2, our goal in this work is to thoroughly compare the security and availability properties of the most important low-latency secure channel establishment protocols: TLS 1.3 over TFO, QUIC over UDP, and QUIC[TLS] over UDP.

4.1.1 Our Contributions

To compare security, we first need to define a general protocol syntax for secure channel establishment and fix a security model for it. Since the only provable security analysis that studies security related to data transmission functionality is [32], we take their *Quick Connections (QC)* protocol definition and *Quick Authenticated and Confidential Channel Establishment (QACCE)* security model as our starting point.

To accommodate protocol syntaxes of TLS 1.3 and QUIC[TLS], we extend the QC protocol to a more general *Multi-Stage Authenticated and Confidential Channel Establishment (msACCE)* protocol, which allows more keys to be set during each session. Then, we extend the *Quick Authenticated and Confidential Channel Establishment (QACCE)* security model [32] to two msACCE security models — msACCE-std and msACCE-pauth — that are general enough for all layered secure channel establishment protocols listed in Table 1.1. The former is fairly standard and is for core cryptographic security, and the latter is novel and is for packet-level security.

Like most security models, we consider a very powerful attacker who can initiate communications between honest parties, can intercept, inject, drop, or modify the exchanged packets, and can adaptively learn parties' stage keys or adaptively corrupt them to learn their long-term keys and secret states. The attacker can also have prior knowledge of the exchanged data. However, the attacker should not be able to prevent clients from establishing final session keys without noticing the attacker's involvement (Server Authentication) or using these keys to achieve a secure channel with data privacy and integrity (Channel Security). These standard security goals are captured by our first model.

For the second model that deals with packet-level availability attacks, we first follow QACCE [32] to consider IP-spoofing prevention (also known as address validation) and further extend it to additionally capture IP-spoofing attacks in the full connections. Then, we design several novel notions for packet-level authentication as follows.

First, we define Header Integrity to capture the integrity of the whole unencrypted packet header. (Note that previous models like QACCE only cover the header integrity implied by the authenticity security of the underlying authenticated encryption scheme.) To enable fine-grained security analyses and comparisons, we split the above notion into two related ones, Key Exchange (KE) Header Integrity and Secure Channel (SC) Header Integrity, which capture header integrity during the key exchange phase and secure channel phase respectively. Furthermore, we define the notion of KE Payload Integrity to cover availability attacks that modify the payloads of packets sent during key exchange. We note that unlike the availability attacks shown in [32], successful attacks under our new notions do not affect the client's session key establishment and therefore are harder or impossible to detect by the client. This makes such attacks more harmful and their treatment more important. Finally, we formalize the new goal of Reset Authentication to deal with attacks forging a reset packet to abruptly terminate an honest party's session.

Equipped with our new models, we study the security and availability functionalities provided by TFO+TLS 1.3, UDP+QUIC, and UDP+QUIC[TLS]. We first confirm that all protocols provably satisfy the standard security notions of Server Authentication and Channel Security given that their building blocks are secure. The results mostly follow from prior works and we just have to argue that they still hold for the extended model.

Similarly, prior results showed that QUIC achieves IP-spoofing prevention and we show that this extends to our stronger notion. As for TFO+TLS 1.3, its IP-spoofing prevention relies on TCP sequence number randomization and TFO's cookie mechanism (but no prior former analysis confirmed its security). We prove that TFO+TLS 1.3 does satisfy this security assuming that the underlying block cipher is a pseudorandom function.

Regarding SC Header Integrity, we show that while UDP+QUIC is secure, TFO+TLS 1.3, on the other hand, is insecure because it allows header-only packets to be sent in the secure channel phases and does not authenticate the TCP headers of encrypted packets. This theoretical result captures practical availability attacks that the networking community has been slowly uncovering via manual investigation over the last 30 years [82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95], such as TCP flow control manipulation, TCP acknowledgment injection, etc.

We next show that neither protocol satisfies KE Header Integrity. For TFO+TLS 1.3 this result leads to a TFO cookie removal attack that we discover, which allows the attacker to undermine the whole benefit of TFO. Then, we show that UDP+QUIC is not secure in the sense of KE Payload Integrity. This leads to a new availability attack that we call ServerReject Triggering. Note that unlike the QUIC attacks (e.g., server config replay attack, connection ID manipulation attack, etc.) discovered in [32], ServerReject Triggering is harder to detect and more harmful in this sense. We show that TFO+TLS 1.3, on the other hand, achieves KE Payload Integrity.

We further show that neither TFO+TLS 1.3 nor UDP+QUIC provide Reset Authentication, justifying the TCP Reset attack [94] relevant for TFO+TLS 1.3 and the PublicReset attack for UDP+QUIC. For completeness, we recall the results from [32, 11] showing that neither protocol provides forward secrecy for the keys encrypting 0-RTT data and that this data can be replayed.

We finally show that the new UDP+QUIC[TLS] protocol achieves the strongest security of three designs. While formally it does not provide KE Payload Integrity, the related

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4.1: TCP header. [96]

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Length             |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4.2: UDP header. [97]

attacks can also happen in TFO+TLS 1.3 in a similar way, while the latter satisfies KE Payload Integrity mainly because its availability functionalities are all carried in its protocol headers rather than payloads. More importantly, UDP+QUIC[TLS] is the only protocol that guarantees Reset Authentication (based on the unpredictability of its reset tokens).

Our results are summarized in Table 1.2 in Section 4.4. Even though QUIC may not be able to sustain the competition in the long run despite stronger security, we hope our models will help protocol designers and practitioners better understand the important security aspects of novel secure channel establishment protocols.

4.2 Background

Network protocols are designed and implemented following a layered network stack model where each layer has its own functionality, defines an interface for use by higher layers, and relies only on the properties of lower layers. In this work, we are concerned with three layers: network, represented by the IP protocol; transport, represented by UDP and TCP with the Fast Open optimization (TFO); and application, represented by TLS or QUIC.

4.2.1 TLS 1.3 over TFO

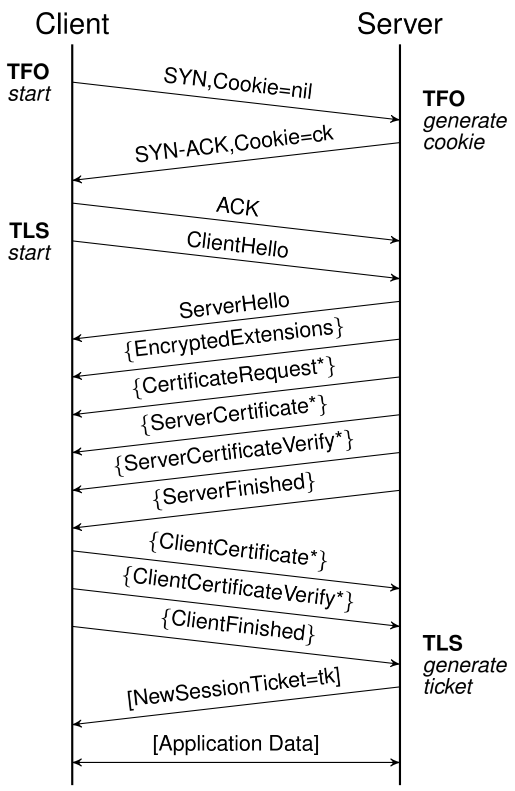
TCP Fast Open. TCP Fast Open (TFO) is an optimization to the TCP protocol. TCP itself provides the following services to an application (or higher protocol): (1) reliability, (2) ordered delivery, (3) flow control, and (4) congestion control. It is connection-oriented and consists of three phases: connection establishment, data transfer, and connection teardown. TCP relies on control information from its header to implement this functionality. For example, as shown in Figure 4.1, control bits specify what type of packets are sent over the network, which determines whether the packets are establishing a new connection, sending data, acknowledging data, or tearing down the connection.

The disadvantage of layering protocols is that higher level protocols have no control over the internal mechanics of lower level protocols and can interact with them only through defined interfaces. A protocol using standard TCP for transferring data needs to wait for connection establishment at the TCP layer to complete before it receives notification of a new connection and can begin its own processing and data transfer.

The TFO optimization introduces a simple modification to the TCP connection establishment handshake to reduce the 1-RTT connection establishment latency of TCP and allow for 0-RTT handshakes, so that data transmission may begin immediately. TFO fulfills the same design goals mentioned for TCP above, assuming the connection is established correctly.

The mechanism through which 0-RTT is achieved is a cookie that is obtained by the client first time it communicates with a server and cached for later uses. This cookie is intended to prevent replay attacks while avoiding the need for servers to keep expensive state. It is generated by the server, authenticates client IP address, and has a limited lifetime. Generation and verification have low overhead.

Cookies are sent in the TFO option field in SYN packets. The first two message exchanges in Figure 4.3 left, show how a cookie is obtained. The client requests a cookie by



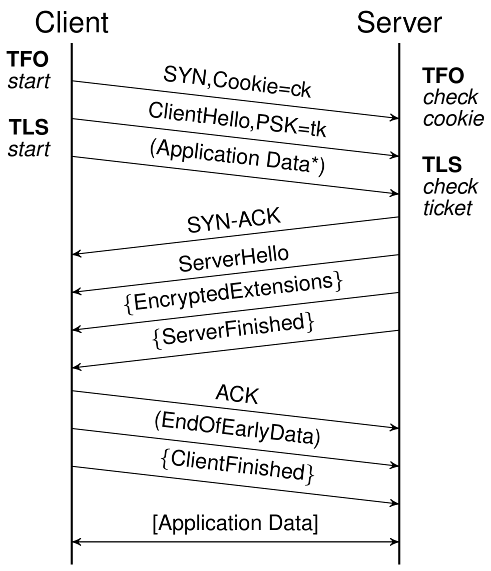


Figure 4.3: TFO+TLS 1.3 (EC) DHE 2-RTT full handshake (left) and TFO+TLS 1.3 PSK-(EC) DHE 0-RTT resumption handshake (right). * indicates optional messages. () indicates messages protected using the 0-RTT keys derived from a pre-shared key. {} and [] indicate messages protected with initial and final keys.

using the TFO option in the SYN with the cookie field set to 0, indicating that it would like to use TFO. The server generates an appropriate cookie and places it in the TFO option field of the SYN-ACK. The client caches this cookie for subsequent connections to this server. If a cookie was not provided, the client instead caches the negative response, indicating that TFO connections should not be tried to this server, for some time.

In subsequent connections to this server (first message in Figure 4.3 right), the client places its cached TFO cookie in the TFO option in the SYN packet. The client is also allowed to send 0-RTT data in the remainder of the SYN packet. This might be an HTTP GET request or a TLS `ClientHello` message. When the server receives the SYN, it will validate the cookie. If the cookie is valid, it responds with a SYN-ACK acknowledging the 0-RTT data and a response to the 0-RTT data. If the cookie is invalid (expired or otherwise), a full handshake is required and any initial data ignored.

TLS 1.3. TLS provides confidentiality, authentication, and integrity of communication over a secure channel between a client and a server. This is accomplished in two phases – the handshake and the record protocol. The handshake sets up appropriate parameters for the record protocol to achieve these three goals. These include parameters like the cipher suite to use and the shared secret key. Unfortunately, the handshake in TLS 1.2 takes 2-RTTs to complete. Additionally, the naive layering of TLS 1.2 over TCP, as traditionally used for HTTPS, would require a full 3-RTTs before the HTTP request could be sent. Fortunately, the recently standardized TLS 1.3 [5] provides many improvements over TLS 1.2. Most relevant for our purposes, it enables 0-RTT handshakes at the TLS level.

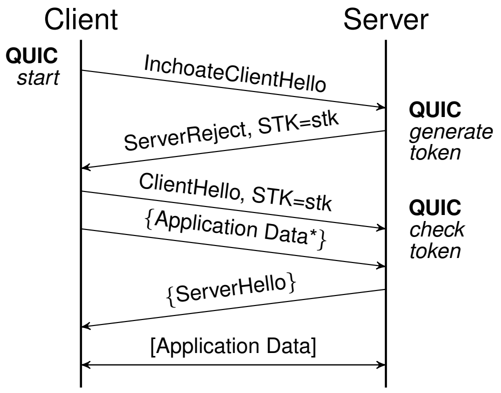
In a TLS 1.3 full connection (see Figure 4.3 left, fourth message), the client begins by sending a `ClientHello` message containing a list of ciphersuites the client is willing to use with key shares for each and optional extensions. The server responds with a `ServerHello` message containing the ciphersuite to use and its key share. At this point, an initial encryption key is derived and all future messages are encrypted. The server also sends an `EncryptedExtensions` message containing any extension data, a `CertificateRequest` message if doing client authentication, a `ServerCertificate` message containing the server's certificate, a `ServerCertificateVerify` message containing a signature over the handshake with the private key corresponding to the server's certificate, and a `ServerFinished` message containing an HMAC of all messages in the handshake. The client receives these messages, verifies their contents, and responds with `ClientCertificate` and `ClientCertificateVerify` messages if doing client authentication before finishing with a `ClientFinished` message containing an HMAC of all messages in the handshake. At this point, a final encryption key is derived and used for encrypting all future messages. If the server supports 0-RTT connections, one final handshake message, the `NewSessionTicket` message, will be sent by the server to provide the client with an opaque session ticket to be used in a resumption session.

In later TLS 1.3 resumption connections to this server, the client uses the session ticket established in the prior full connection to do a 0-RTT connection. In this case, the client sends a `ClientHello` message indicating a pre-shared-key ciphersuite, a ciphersuite to use for the final key, and the cached session ticket. The client can then derive an encryption key and begin sending 0-RTT data. The server will verify the session ticket, use it to establish the same encryption key, and send a `ServerHello` message containing the ciphersuite to use and its final key share. At this point, an initial encryption key is derived and all future messages are encrypted. The server also sends an `EncryptedExtensions` message containing any extension data and a `ServerFinished` message containing an HMAC of all messages in the handshake. The client receives these messages, verifies their contents, and responds with an `EndOfEarlyData` message and a `ClientFinished` message containing an HMAC of all messages in the handshake. At this point, a final encryption key is derived and used for encrypting all future messages.

TLS 1.3 over TFO. TLS assumes that lower layers provide reliable, in-order delivery of TLS messages. As a result, TLS is usually layered on top of TCP, which provides these properties. This usually results in a delay for the TCP handshake followed by a delay for the TLS handshake. This is obviously undesirable. However, the combination of TLS 1.3 and TCP Fast Open enables true 0-RTT connections.

In a full connection to a TFO+TLS 1.3 server, the client requests a TFO cookie in the TCP SYN and then does a full TLS 1.3 handshake once the TCP connection completes. This takes 3-RTTs (see Figure 4.3 left), but provides a cached TFO cookie and cached TLS session ticket.

In subsequent resumption connections to this server, the client can use the TFO cookie to establish a 0-RTT TCP connection and include the TLS 1.3 `ClientHello` message in the SYN packet. The TLS `ClientHello` message can use the cached TLS session ticket to perform a 0-RTT resumption handshake. Thus, the TCP and TLS 1.3 connections are



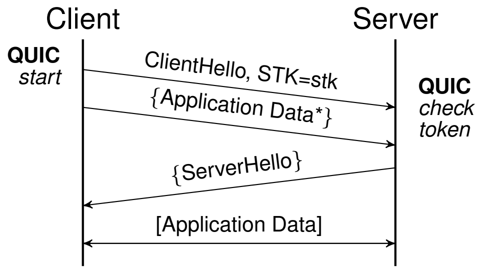


Figure 4.4: QUIC 1-RTT full handshake (left) and UDP+QUIC 0-RTT resumption handshake (right). * indicates optional messages. {} and [] indicate messages protected with initial and final keys.

established at the same time, as shown in Figure 4.3 right.

4.2.2 QUIC over UDP

UDP. UDP [97] is an extremely simple transport protocol providing unreliable datagram delivery, the ability to multiplex data between multiple applications, and an optional checksum. A UDP sender simply wraps the message to be sent with a UDP header (see Figure 4.2) and the receiver unwraps the message and delivers it to the application, after possibly verifying the checksum. No other processing is performed.

UDP has been typically used for applications where low latency is crucial, like video gaming and real-time streaming video. As a result, it can traverse NAT devices and firewalls that often block unknown or rare protocols.

QUIC. Quick UDP Internet Connections (QUIC) is a transport protocol developed by Google and implemented by Chrome and Google servers since 2013 [24]. It now provides service for the majority of requests by Chrome to Google properties [98]. QUIC's goal was to provide secure communication comparable with TLS while achieving reduced connection setup latency compared to traditional TCP+TLS 1.2. To do so, it provides the following services to applications: (1) reliability, (2) in order delivery, (3) flow control,

(4) congestion control, (5) data confidentiality, and (6) data authenticity. For repeated connections to the same server it also provides (7) 0-RTT connections, enabling useful data to be sent in the first round trip. In short, QUIC provides a very similar set of services to TFO+TLS 1.3.

Instead of modifying TCP to enable 0-RTT connection establishment, QUIC replaces TCP entirely, using UDP to provide application multiplexing and enabling it to traverse the widest possible swath of the Internet. QUIC then provides all other guarantees itself.

QUIC packets contain a public header and a set of frames that are encrypted and authenticated after initial connection setup. The header contains a set of public flags, a unique 64-bit connection identifier referred to as `cid`, and a variable length packet number. All other protocol information is carried in control and stream (data) frames that are encrypted and authenticated.

To provide 0-RTT, QUIC caches important information about the server that will enable the client to determine the encryption key to be used for each new connection. As shown in Figure 4.4 left, the first time a client contacts a given server it has no cached information, so it sends an empty (Inchoate) `ClientHello` message. The server responds with a `ServerReject` message containing the server's certificate and three pieces of information for the client to cache. The first of these is an object called an `scfg`, or server configuration. The `scfg` contains a variety of information about the server, including a Diffie-Hellman share from the server, supported encryption and signing algorithms, and flow control parameters. This `scfg` has a defined lifetime and is signed by the server's private key to enable authentication using the server's certificate. Along with the `scfg`, the server sends the client a source-address token or `stk`. The `stk` is used to prevent IP spoofing. It contains an encrypted version of the client's IP address and a timestamp.

With this cached information, a client can establish an encrypted connection with the server. It first ensures that the `scfg` is correctly signed by the server's certificate which is valid and then sends a `ClientHello` indicating the `scfg` its using, the `stk` value it has cached, a Diffie-Hellman share for the client, and a client nonce. After sending the `ClientHello`, the client can create an initial encryption key and send additional encrypted `Application Data` packets. In fact, to take advantage of the 0-RTT connection establishment it must do so. When the server receives the `ClientHello` message, it validates the `stk` and client nonce parameters and creates the same encryption key using the server share from the `scfg` and the client's share from the `ClientHello` message.

At this point, both client and server have established the connection and setup encryption keys and all further communication between the parties is encrypted. However, the connection is not forward secure yet, meaning that compromising the server would compromise all previous communication because the server's Diffie-Hellman share is the same for all connections using the same `scfg`. To provide forward secrecy for all data after the first RTT, the server sends a `ServerHello` message after receiving the client's `ClientHello` which contains a newly generated Diffie-Hellman share. Once the client receives this message, client and server derive and begin using the new forward secure encryption key.

For the client that has connected to a server before, it can instead initiate a resumption connection. This consists of only the last two steps of a full connection, sending the `ClientHello` and `ServerHello` messages as shown in Figure 4.4 right.

4.2.3 QUIC with TLS 1.3 Key Exchange

A new version of QUIC [27], which also supports 0-RTT, describes several improvements of the previous design. The most important change is replacing QUIC's key exchange with the one from TLS 1.3, as specified in the latest Internet draft [28]. We provide more details (e.g., the new stateless reset feature) of this new QUIC (denoted by QUIC[TLS]) in Section 4.4.

4.3 Multi-Stage Authenticated and Confidential Channel Establishment

In this section, we define the syntax and two security models for *Multi-Stage Authenticated and Confidential Channel Establishment (msACCE)* protocols.

For simplicity, we assume the public keys used in our analysis are supported by a *public key infrastructure (PKI)* and do not consider certificates or certificate checks explicitly. In other words, we assume each public key is certified and bound to the corresponding party's identity.

4.3.1 Protocol Syntax

Our msACCE protocol is an extension to the *Quick Connection (QC)* protocol proposed by Lychev *et al.* [32] and the *Multi-Stage Key Exchange (MSKE)* protocol proposed by Fischlin and Günther [31] (and further developed by [6, 9, 10, 11]). Even though the authors of [32] claimed their QC protocol syntax to be general, TLS 1.3 does not fit it well because TLS 1.3 has two initial keys and one final key in 0-RTT resumption while QC captures only one initial key. On the other hand, the MSKE protocol and its extensions focus only on the key exchange phases.

Our msACCE protocol syntax inherits many parts of the QC protocol syntax but extends it to a multi-stage structure and additionally covers session resumptions (explicitly, unlike QC), session resets, and header-only packets exchanged in secure channel phases. The detailed protocol syntax is defined below.

A msACCE protocol is an interactive protocol between a client and a server. They establish keys in one or more stages and exchange messages encrypted and decrypted with these keys. Messages are exchanged via *packets*. A packet consists of source and destination IP addresses[1] $\mathsf{IP_s}, \mathsf{IP_d} \in \{0,1\}^{32} \cup \{0,1\}^{64}$, a header, and a payload. Each party P has a unique IP address IP_P.

[1]For the network-layer protocols, we only consider the Internet Protocol and its IP address header fields because our model mainly focuses on the application and transport layers and additionally only captures the IP-spoofing attack.

The protocol is associated with the security parameter $\lambda \in \mathbb{N}_+$, a key generation algorithm Kg that takes as input 1^λ and outputs a public and secret key pair, a header space[2] (for transport and application layers) $\mathcal{H} \subseteq \{0,1\}^*$, a payload space $\mathcal{PD} \subseteq \{0,1\}^*$, header and payload spaces $\mathcal{H}_{\mathsf{rst}} \subseteq \mathcal{H}, \mathcal{PD}_{\mathsf{rst}} \subseteq \mathcal{PD}$ for reset packets (described later), a resumption state space $\mathcal{RS} \subseteq \{0,1\}^*$, a stateful AEAD scheme[3] $\mathsf{sAEAD} = (\mathsf{sGen}, \mathsf{sEnc}, \mathsf{sDec})$ (with a key space $\mathcal{K} = \{0,1\}^\lambda$, a message space $\mathcal{M} \subseteq \{0,1\}^*$, an associated data space $\mathcal{AD} \subseteq \{0,1\}^*$, and a state space $\mathcal{ST} \subseteq \{0,1\}^*$), *disjoint*[4] message spaces $\mathcal{M}_{\mathsf{KE}}, \mathcal{M}_{\mathsf{SC}}, \mathcal{M}_{\mathsf{pRST}} \subseteq \mathcal{M}$ with $\mathcal{M}_{\mathsf{KE}}, \mathcal{M}_{\mathsf{SC}}$ for messages encrypted during key exchange and secure channel phases respectively and $\mathcal{M}_{\mathsf{pRST}}$ for pre-reset messages (described later) encrypted in a secure channel phase, and a server configuration generation function $\mathsf{scfg\_gen}$ described below.

The protocol's execution is associated with the universal notion of time divided into discrete periods $\tau_1, \tau_2, \ldots$. During its execution, both parties can keep states that are initialized to the empty string ε. In the beginning of each time period, the protocol may periodically update each server's configuration state `scfg` with $\mathsf{scfg\_gen}$ (which takes as input 1^λ, a server secret key, and a time period, then outputs a server configuration state). Otherwise, $\mathsf{scfg\_gen}$ is undefined and without loss of generality the protocol is executed within a single time period.

A *reset* packet enables a sender, who lost its session state due to some error condition (e.g., server reboots, denial-of-service attacks, etc.), to abruptly terminate a session with the receiver. A *pre-reset* message (e.g., a reset token in QUIC[TLS]) is sent to the receiver in a secure channel phase[5] before the sender loses its state in order to authenticate the sender's

[2]Some protocol header fields (e.g., port numbers, checksums, etc.) can be excluded if they are not the focus of the security analysis.

[3]To fit TLS 1.3's encryption scheme, unlike QACCE we model QUIC's encryption scheme as a more general stateful AEAD scheme rather than a nonce-based one.

[4]Disjointness is a reasonable assumption as practical protocols (such as those in Table 1.1) enforce different leading bits for different types of messages.

[5]A pre-reset message can also be carried within an *encrypted* key exchange packet. We consider it encrypted as a separate secure channel packet to get a clean packet-authentication security model described later.

reset packet. Each session has *at most one* pre-reset message for each party. A *non-reset* packet is not a reset packet. A *header-only* packet has no payload.

We say a party *rejects* a packet if its processing the packet leads to an error (defined according to the protocol), and *accepts* it otherwise.

The protocol has two modes, *full* and *resumption*. Its corresponding executions are referred to as the full and resumption sessions. Each resumption session is associated with a *single* previous full session and we say the resumption session *resumes* its associated full session. In the beginning of a full or resumption session, each party takes as input a list of messages[6] $\mathcal{M}^{\mathsf{send}} = (M_1, \ldots, M_l), M_i \in \mathcal{M}_{\mathsf{SC}}, l \in \mathbb{N}$ (where the total message length $|\mathcal{M}^{\mathsf{send}}|$ is polynomial in λ and $\mathcal{M}^{\mathsf{send}}$ can be empty) as well as the other party's IP address. In a full session, the server takes as input its associated public and secret key pair (generated by running $\mathsf{Kg}(1^\lambda)$) and the client takes the server's public key as input. In a resumption session, each party additionally takes as input its own resumption state $rs \in \mathcal{RS}$ (set in the associated full session). In either case, the client sends the first packet to start the session.

A D-stage msACCE protocol consists of $D \in \mathbb{N}_+$ successive stages and each stage, e.g., the d-th ($d \in [D]$) stage, consists of one or two phases described as follows:

1) Key Exchange. At the end of this phase each party sets its d-th stage key $k^d = (k_c^d, k_s^d)$. At most one of k_c^d and k_s^d can be $\perp$, i.e., unused.[7] If this is the final stage in a full session, each party can send additional messages[8] in $\mathcal{M}_{\mathsf{KE}}$ encrypted with k^d and by the end of this phase each party sets its own resumption state.

2) Secure Channel. This phase is mandatory for the final stage but optional for other stages. In this phase, the parties can exchange messages from their input lists as well as pre-reset messages, encrypted and decrypted using the associated stateful AEAD scheme with k^d. The client uses k_c^d to encrypt and the server uses it to decrypt, whereas the server

[6]For simplicity, we consider transportation of *atomic* messages rather than a data *stream* that can be modeled as a stream-based channel [99] and later extended to capture multiplexing [100].

[7]This captures the case where a 0-RTT key only consists of a client encryption key while the server encryption key does not exist.

[8]This captures the post-handshake key exchange messages that are used for session resumption, post-handshake authentication, key update, etc.

uses k_s^d to encrypt and the client uses it to decrypt. They may also send reset or header-only packets. At the end of this phase, each party outputs a list of received messages (which may be empty) $\mathcal{M}_i^{\mathsf{recv}} = (M'_1, \ldots, M'_{l'_i})$, $l'_i \in \mathbb{N}$, $M'_i \in \mathcal{M}_{\mathsf{SC}}$.

Each message exchanged between the parties must belong to some unique phase at some unique stage. One stage's second phase and the next stage's first phase may overlap, and the two phases in the final stage may also overlap. We call the final stage key the *session* key and the other stage keys the *interim* keys.

Correctness. Consider a client and a server running a D-stage msACCE protocol in either mode without sending any reset packet. Each party's input message list $\mathcal{M}^{\mathsf{send}}$, in which the messages are sent among D stages according to any partitioning $\mathcal{M}^{\mathsf{send}} = \mathcal{M}_1^{\mathsf{send}}, \ldots, \mathcal{M}_D^{\mathsf{send}}$, is equal to the other party's total output message list $\mathcal{M}^{\mathsf{recv}} = \mathcal{M}_1^{\mathsf{recv}}, \ldots, \mathcal{M}_D^{\mathsf{recv}}$, in which the message order is preserved. Each party terminates its session upon receiving the other party's reset packet.

REMARK. With our more general protocol syntax, the ACCE [4] and QC [32] protocols can be classified into 1-stage and 2-stage msACCE protocols respectively.

4.3.2 Security Models

We propose two security models respectively for basic authenticated and confidential channel security and packet authentication. Our models do not consider the key exchange and secure channel phases independently, as was the case for some previous QUIC and TLS 1.3 security analyses [31, 6, 9, 10, 11], because QUIC's key exchange and secure channel phases are inherently inseparable and the TLS 1.3 full handshake does not fit into a composability framework, as discussed in [32, 9].

1) msACCE Standard Security Model:

In this msACCE standard (msACCE-std) security model, we consider the standard security goals such as server authentication[9] and channel security (which captures data pri-

[9]Our msACCE-std model focuses on the most common server authentication, but can be extended to

vacy and integrity) for msACCE protocols. Our msACCE-std model is very similar to the standard security portion of the QACCE model [32], but extends it to capture more (rather than two) stages and use a more general stateful encryption scheme to fit both TLS 1.3 and QUIC.

Like QACCE and other previous models, we consider a very powerful adversary who can control communications between honest parties, can adaptively learn their stage keys, and can adaptively corrupt servers to learn their long-term keys and secret states.

Our detailed security model is defined below.

Protocol Entities. The set of parties $\mathcal{P}$ consists of two disjoint type of parties: clients $\mathcal{C}$ and servers $\mathcal{S}$, i.e., $|\mathcal{P}| = |\mathcal{C}| + |\mathcal{S}|$.

Session Oracles. To capture multiple sequential and parallel protocol executions, each party $P \in \mathcal{P}$ is associated with a set of session oracles $\pi_P^1, \pi_P^2, \ldots$, where π_P^i models P executing a protocol instance in session $i \in \mathbb{N}_+$.

Matching Conversations. As part of the security model, *matching conversations* are used to model entity authentication, session key confirmation, and handshake integrity. A client (resp. server) oracle has a matching conversation with a server (resp. client) oracle if and only if both session oracles observe the same[10] *session identifier* `sid` defined according to the protocol specifications and security goals. Note that a msACCE protocol may have two different session identifiers in full and resumption modes, but for simplicity we use the same notation `sid`. Compared to the general definition of matching conversations [101, 4], `sid` is often defined as a *subset* of the whole communication transcript. For instance, QUIC's `sid` in QACCE [32] is defined as the second-round key exchange messages, i.e., `ClientHello` and `ServerHello`, while the first-round messages are excluded to allow

mutual authentication, e.g., as described in [29].

[10]As discussed in [4], two session oracles having matching conversations with each other may not observe the same transcript due to the gap between one oracle sending a message and the other receiving it. We can use *symmetric* session identifiers to define matching conversations because our msACCE-std model focuses only on server authentication and we require session identifiers to exclude, if any, a client oracle's last key exchange message(s) sent immediately before it sets its session key.

for valid but different source-address tokens or signatures. Similarly, TLS 1.2's `sid` in ACCE [29] is defined as the first three key exchange messages, while the rest are excluded to allow for valid but different encrypted `Finished` messages.

Partners. We say a client oracle and a server oracle are each other's *partner* if they observe the same first-stage session identifier $\mathtt{sid}_1$ (i.e., `sid` restricted to the first stage), which intuitively means that they set the first stage key with each other. Note that a client oracle may have more than one partners if $\mathtt{sid}_1$ consists of only message(s) sent from the client oracle, which can be replayed to the same[11] server to establish multiple (identical) first-stage keys. Therefore, a session oracle's partner may not be its final unique communication partner. Instead, the real partner is the session oracle with which the oracle has a matching conversation.

Security Experiments. In the beginning of the experiments, run $\mathsf{Kg}(1^\lambda)$ for all servers to generate the public and secret key pairs and initialize the global states of all parties and the local states of all session oracles. In the beginning of each time period, run $\mathsf{scfg\_gen}$ (if defined) for each server to update its configuration state `scfg`. We assume that both the server oracles and the adversary $\mathcal{A}$ are aware of the current time period. Let $N \in \mathbb{N}_+$ denote the maximum number of msACCE protocol instances for each party and $D \in \mathbb{N}_+$ denote the maximum number of stages in each session. The channel security experiment is associated with an authentication level $al \in [4]$. Each oracle π_P^i at stage d is associated with a random bit $b_P^{i,d} \xleftarrow{\$} \{0,1\}$. Each oracle π_P^i has a global state $\widetilde{m}$ (initialized to $\perp$) that stores its pre-reset message. The adversary $\mathcal{A}$ is given all public keys and the IP addresses associated with all parties and then interacts with the session oracles via the following queries:

• $\mathsf{Connect}(\pi_C^i, S)$, for $C \in \mathcal{C}, S \in \mathcal{S}, i \in [N]$.

This query asks π_C^i to output the first packet that it would send to S in a full session ac-

[11]In practice, 0-RTT replay attacks can be mounted to *different* servers with the same public-secret key pair. However, 0-RTT key exchange message(s) replayed to other servers with different public-secret key pairs will be rejected.

cording to the protocol if π_C^i was not *used* (i.e., as input of previous $\mathsf{Connect}, \mathsf{Resume}, \mathsf{Send}$ queries). This output packet is returned to $\mathcal{A}$ instead of S. After this query, we say S is the *target server* of π_C^i.

This query allows the adversary to ask a specified client oracle to start a full session with a specified server.

• $\mathsf{Resume}(\pi_C^i, S, i')$, for $C \in \mathcal{C}, S \in \mathcal{S}, i, i' \in [N], i' < i$.

This query asks π_C^i to output the first packet that it, taking $\pi_C^{i'}$'s resumption state as input, would send to S in a resumption session according to the protocol and returns this packet to $\mathcal{A}$, if π_C^i was not used and $\pi_C^{i'}$ has set its resumption state in a previous full session with its target server S. Otherwise, it returns $\perp$.

This query allows the adversary to ask a specified client oracle to start a resumption session with a specified server oracle to resume a specified full session between the two parties, if the associated previous client oracle has set its resumption state.

• $\mathsf{Send}(\pi_P^i, pkt)$, for $P \in \mathcal{P}, i \in [N], pkt \in \{0,1\}^*$.

This query sends pkt to π_P^i and returns its response if π_P^i is in a key exchange phase, otherwise, returns $\perp$.

This query allows the adversary to send any packet to a specified session oracle and get its response in a key exchange phase.

• $\mathsf{Reveal}(\pi_P^i, d)$, for $P \in \mathcal{P}, i \in [N], d \in [D]$.

This query returns π_P^i's (perhaps unset) stage-d key k^d. After this query, we say k^d was *revealed*.

This query allows the adversary to learn any stage key of a specified session oracle.

• $\mathsf{Corrupt}(S)$, for $S \in \mathcal{S}$.

This query returns S's secret key and all its current states including its `scfg` and resumption states (for all full sessions involving S) in the current time period. After this query, we say S was *corrupted*.

This query allows the adversary to learn the long-term secret along with all current states of a specified server.

• $\mathsf{Encrypt}(\pi_P^i, d, ad, m_0, m_1)$, for $P \in \mathcal{P}, i \in [N], d \in [D], ad \in \mathcal{AD}, m_0, m_1 \in \mathcal{M}_{\mathsf{SC}} \cup \mathcal{M}_{\mathsf{pRST}} \cup \{\mathtt{rst}\}$.

This query proceeds as follows:

```
1: if $m_0, m_1$ are of different types (i.e., $\mathcal{M}_{\mathsf{SC}}$ or $\mathcal{M}_{\mathsf{pRST}}$ or $\{\mathtt{rst}\}$) or $|m_0| \neq |m_1|$ or $\pi_P^i$ is not in its $d$-th secure channel phase or $k_p^d = \perp$ (where $p = c$ if $P \in \mathcal{C}$ and $p = s$ if $P \in \mathcal{S}$), return $\perp$
2: if $m_0 = m_1 = \mathtt{rst}$, return $\widetilde{m}$
3: if $m_0, m_1 \in \mathcal{M}_{\mathsf{pRST}}$, return $\perp$ if $\widetilde{m} \neq \perp$ or set $\widetilde{m} \leftarrow m_{b_P^{i,d}}$ otherwise
4: (upon setting each encryption stage key, initialize $st_e \in \mathcal{ST}, u \leftarrow 0, sent \leftarrow \varepsilon$)
5: $u \leftarrow u + 1, (sent.ct_u, st'_e) \xleftarrow{\$} \mathsf{sEnc}(k_p^d, ad, m_{b_P^{i,d}}, st_e)$
6: $(sent.ad_u, st_e) \leftarrow (ad, st'_e)$
7: return $sent.ct_u$
```

This query allows the adversary to specify any associated data and any two secure channel or pre-reset messages of the same length, then get the ciphertext of one message determined by $b_P^{i,d}$, a random bit associated with the specified session oracle at the specified stage. This query also stores the pre-reset message and returns it when the input is $\mathtt{rst}$. (Recall that each oracle has at most one pre-reset message, so this query stores only the first pre-reset message and rejects others.)

• $\mathsf{Decrypt}(\pi_P^i, d, ad, ct)$, for $P \in \mathcal{P}, i \in [N], d \in [D], ad \in \mathcal{AD}, ct \in \{0,1\}^*$.

This query proceeds as follows:

```
1: if $\pi_P^i$ is not in its $d$-th secure channel phase or $k_p^d = \perp$ (where $p = c$ if $P \in \mathcal{C}$ and $p = s$ if $P \in \mathcal{S}$), return $\perp$
2: (upon setting each decryption stage key, initialize $st_d \in \mathcal{ST}, v \leftarrow 0, rcvd \leftarrow \varepsilon, \mathtt{outofsync} \leftarrow 0$)
3: $v \leftarrow v + 1, rcvd.ct_v \leftarrow ct, (m, st'_d) \leftarrow \mathsf{sDec}(k_p^d, ad, ct, st_d)$
4: $(rcvd.ad_v, st_d) \leftarrow (ad, st'_d)$
```

```
5: if $m \notin \mathcal{M}_{\mathsf{SC}} \cup \mathcal{M}_{\mathsf{pRST}}$, set $m \leftarrow \bot$
6: if $(al = 4) \wedge \mathsf{cond}_4$ or $(al \leq 3) \wedge (m \neq \bot) \wedge \mathsf{cond}_{al}$,
       set $\mathtt{outofsync} \leftarrow 1$
7: if $\mathtt{outofsync} = 1$, return $b_P^{i,d}$, otherwise, return $\bot$
```

This query allows the adversary to specify any associated data and any ciphertext to be decrypted by the *partner(s)* of the specified session oracle at the specified stage, then get the secret bit $b_P^{i,d}$ if and only if this query is "out-of-sync", otherwise, it still gets $\bot$ (to avoid trivial wins). The "out-of-sync" condition (see line 6) captures different authentication levels. For conciseness, we list only the authentication conditions for level 1 and 4 (refer to [46] for level 2 and 3) as follows:

$$\mathsf{cond}_1 = (\nexists w : (ct = sent.ct_w) \wedge (ad = sent.ad_w))$$

$$\mathsf{cond}_4 = (u < v) \vee (ct \neq sent.ct_v) \vee (ad \neq sent.ad_v)$$

Note that cond_1 corresponds to the lowest authentication level (e.g., for the stateful AEAD scheme in QUIC) that only guarantees no forgeries, while cond_4 corresponds to the highest authentication level (e.g., for the stateful AEAD scheme in TLS 1.3) that prevents forgeries, replays, reordering, and dropping.

Advantage Measures. An adversary $\mathcal{A}$ against a msACCE protocol Π in msACCE-std has the following advantage measures.

• *Server Authentication.* We define $\mathbf{Adv}_{\Pi}^{\mathsf{s\text{-}auth}}(\mathcal{A})$ as the probability that there exist a client oracle π_C^i and its target server S such that the following holds:

1. π_C^i has set its session key;
2. S was not corrupted before π_C^i set its session key;
3. No interim keys of π_C^i or its partner(s) were revealed;[12]

[12]More precisely, we allow revealing other stage keys (if any), except the first stage key, of a π_C^i's partner which observes the same session identifier at only the first stage but not the next one, because such a partner's key exchange message is never received by π_C^i. Similar condition relaxation also holds for our other security notions.

4. There is no unique server oracle π_S^j with which π_C^i has a matching conversation.

The above captures the attacks in which the adversary impersonates a server to make the client mistakenly believe that it shares the session key with the server.

• *(level-al) Channel Security.* We define $\mathbf{Adv}_{\Pi}^{\text{cs-}al}(\mathcal{A})$ as $|2\Pr[b_P^{i,d} = b'] - 1|$, where $al \in [4]$ is a specified authentication level and (P, i, d, b') is output by $\mathcal{A}$, such that the following holds:

1. If $P = S \in \mathcal{S}$, π_S^i has a matching conversation with a client oracle π_C^j; if $P = C \in \mathcal{C}$, denote S as π_C^i's target server;

2. S was not corrupted before π_P^i set its last stage key; If *forward secrecy* is not required for the d-th stage keys, S was not corrupted in the same time period associated with π_P^i;

3. No stage keys of π_P^i or its partner(s) were revealed;

4. If two different pre-reset messages were queried in the d-th stage, later no $\mathsf{Encrypt}(\pi_P^i, \cdot, \cdot, \mathtt{rst}, \mathtt{rst})$ queries were made.

The above captures the attacks in which the adversary compromises the privacy or integrity of secure channel messages without revealing stage keys or revealing the hidden pre-reset message or corrupting the server before the client set its last stage key (which may not be the session key). If the stage key at the target stage is not supposed to provide forward secrecy, the adversary is further restricted not to corrupt the server during the same associated time period of the target session.

2) msACCE Packet-Authentication Security Model:

In this msACCE packet-authentication (msACCE-pauth) security model, we consider security goals related to packet authentication beyond those captured by the msACCE-std model. Note that msACCE-std essentially focuses only on the packet fields in the application layer, while msACCE-pauth further covers transport-layer headers and IP addresses.

First, we consider IP spoofing prevention (a.k.a. source authentication) as with the QACCE model, but, as illustrated later, generalize one of the QACCE queries to additionally capture IP spoofing attacks in the full sessions. Then, more importantly, we define four novel packet-level security notions (elaborated later): *KE Header Integrity*, *KE Payload Integrity*, *SC Header Integrity*, and *Reset Authentication*, which enable a comprehensive and fine-grained security analysis of layered protocols.

In particular, KE Header and Payload Integrity respectively capture the header and payload integrity of key exchange packets. Such security issues have not been investigated before and, as we show later, lead to new availability attacks for both TFO+TLS 1.3 and UDP+QUIC. Furthermore, we employ SC Header Integrity to capture the header integrity of non-reset packets in secure channel phases. Note that, unlike the availability attacks shown in [32], successful attacks breaking our security notions are *harder or impossible to detect* by the client as they do not affect the client's session key establishment, so they are more harmful in this sense. Finally, our model captures malicious *undetectable* session resets in a secure channel phase with Reset Authentication.

As with the msACCE-std model, msACCE-pauth captures multiple stages and considers a very powerful adversary. It also inherits the same definitions of protocol entities, session oracles, matching conversations, and partners.

Security Experiments. Consider the same experiment setups as in msACCE-std, except that no random bit $b_P^{i,d}$ is needed. The adversary $\mathcal{A}$ is given all the public parameters and interacts with the session oracles via the same $\mathsf{Connect}$, Resume, Send, Reveal, $\mathsf{Corrupt}$ queries as in the msACCE-std model[13], as well as the following:

• $\mathsf{Connprivate}(\pi_C^i, \pi_S^j, \mathtt{cmp})$, for $C \in \mathcal{C}$, $S \in \mathcal{S}$, $i, j \in [N]$, $\mathtt{cmp} \in \{0, 1\}$.

This query always returns $\perp$. If $\mathtt{cmp} = 1$, π_C^i and π_S^j establish a *complete* full session privately without showing their communication to the adversary. If $\mathtt{cmp} = 0$, π_C^i and π_S^j establish a *partial* full session privately such that the last packet sent from π_C^i right before

[13]Note that $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$ queries are not needed because msACCE-pauth does not consider data privacy explicitly.

π_S^j sets its first stage key is blocked.

This query allows the adversary to establish a complete or partial full session between any client and server oracles without observing their communication. By taking an additional flag `cmp` as input, this query extends the QACCE Connprivate query [32] to model IP-spoofing attacks happening in both *full* and resumption sessions.

• $\mathsf{Pack}(\pi_P^i, ad, m)$, for $P \in \mathcal{P}, i \in [N], ad \in \mathcal{AD}, m \in \mathcal{M}_{\mathsf{SC}} \cup \mathcal{M}_{\mathsf{pRST}} \cup \{\mathtt{prst}, \mathtt{rst}\}$.
This query returns $\perp$ if π_P^i is not in a secure channel phase. If $m \in \mathcal{M}_{\mathsf{pRST}}$, it asks π_P^i to set its pre-reset message equal to m (which may fail). If $m = \mathtt{prst}$, it asks π_P^i to generate its own pre-reset message (hidden from the adversary). (Recall that each oracle has at most one pre-reset message, so at most one of the above queries is successful for π_P^i.) If $m \in \mathcal{M}_{\mathsf{SC}} \cup \mathcal{M}_{\mathsf{pRST}} \cup \{\mathtt{prst}\}$, this query then asks π_P^i to output the packet that it would send to its partner(s) for the specified associated data ad and message m (which are useless if $m = \mathtt{prst}$) according to the protocol, then returns this packet. If $m = \mathtt{rst}$, this query asks π_P^i to output its reset packet (if any) and returns it.

This query allows the adversary to specify any associated data and any message in a secure channel phase, then get the packet output by the specified session oracle. The adversary can also ask the specified session oracle to set a specified pre-reset message or get its reset packet.

• $\mathsf{Deliver}(\pi_P^i, pkt)$, for $P \in \mathcal{P}, i \in [N], pkt \in \{0,1\}^*$.
This query delivers pkt to π_P^i and returns its response if π_P^i is in a secure channel phase, otherwise, returns $\perp$.

This query allows the adversary to deliver any packet to a specified session oracle and get its response in a secure channel phase.

Advantage Measures. An adversary $\mathcal{A}$ against a msACCE protocol Π in msACCE-pauth has the following associated advantage measures.

• *IP-Spoofing Prevention.* We define $\mathbf{Adv}_{\Pi}^{\mathsf{ipsp}}(\mathcal{A})$ as the probability that there exist a client oracle π_C^i and a server oracle π_S^j such that the following holds:

1. π_S^j has set its first stage key right after a $\mathsf{Send}(\pi_S^j, (\mathrm{IP}_C, \mathrm{IP}_S, \cdot, \cdot))$ query;

2. S was not corrupted before π_S^j set its first stage key;

3. The only allowed queries concerning both C and S in the time period associated with π_S^j are:

   - $\mathsf{Connprivate}(\pi_C^x, \pi_S^y, \cdot)$ for any $x, y \in [N]$, and
   - $\mathsf{Send}(\pi_S^y, (\mathrm{IP}_C, \mathrm{IP}_S, \cdot, \cdot))$ for any $y \in [N]$, where $(\mathrm{IP}_C, \mathrm{IP}_S, \cdot, \cdot)$ is the last packet received by π_S^y right before it sets its first stage key.

The above captures the attacks in which the adversary fools a server into accepting a spurious connection request seemingly from an impersonated client, without observing any previous communication between the client and server in the same time period.

• *KE Header Integrity.* We define $\mathbf{Adv}_{\Pi}^{\mathsf{int\text{-}keh}}(\mathcal{A})$ as the probability that there exist a client oracle π_C^i and a server oracle π_S^j such that the following holds:

1. π_C^i has set its session key and has a matching conversation with π_S^j;

2. S was not corrupted before π_C^i set its session key;

3. No interim keys of π_C^i or its partner(s) were revealed;

4. In a key exchange phase before π_C^i set its session key, π_C^i (resp. π_S^j) accepted a packet with a new header that was not output by π_S^j (resp. π_C^i).

The above captures the attacks in which the adversary modifies the protocol header of a key exchange packet of the communicating parties without affecting the client setting its session key. In the above definition, we assume that a client sets its session key *immediately* after sending its last key exchange packet(s) (if any). Then, a forged packet that leads to a successful attack cannot be any of these last packet(s), which have not yet been sent to the server. The same assumption is made for KE Payload Integrity defined below.

• *KE Payload Integrity.* We define $\mathbf{Adv}_{\Pi}^{\textsf{int-kep}}(\mathcal{A})$ as the probability that there exist a client oracle π_C^i and a server oracle π_S^j such that the same (1)$\sim$(3) conditions as in the above KE Header Integrity notion hold and the following holds:

4. In a key exchange phase before π_C^i set its session key, π_C^i (resp. π_S^j) accepted a packet with a new payload that was not output by π_S^j (resp. π_C^i).

The above captures the attacks in which the adversary modifies the payload of a key exchange packet of the communicating parties without affecting the client setting its session key.

• *SC Header Integrity.* We define $\mathbf{Adv}_{\Pi}^{\textsf{int-h}}(\mathcal{A})$ as the probability that $\mathcal{A}$ outputs (P, i, d) such that the same (1)$\sim$(3) conditions as in the Channel Security notion hold and the following holds:

4. In the secure channel phase of the d-th stage, π_P^i accepted a non-reset packet with a new header that was not output by its partner(s) (via Pack queries), or π_P^i accepted a non-reset header-only packet.

The above captures the attacks in which the adversary creates a valid non-reset secure channel packet by forging the protocol header without breaking any Channel Security conditions. Note that in the above security notion an invalid header forgery is detected *immediately* after the malicious packet is received and processed, while the detection of invalid packet forgeries in a key exchange phase (e.g., for plaintext packets) can be *delayed* to the point when the client sets its session key, according to the definitions of KE Header and Payload Integrity.

• *Reset Authentication.* We define $\mathbf{Adv}_{\Pi}^{\textsf{rst-auth}}(\mathcal{A})$ as the probability that $\mathcal{A}$ outputs (P, i, d) such that the same (1)$\sim$(3) conditions as in the Channel Security notion hold and the following holds:

4. In the secure channel of the d-th stage, π_P^i accepted a packet output by a

$\mathsf{Pack}(\cdot, \cdot, \mathtt{prst})$ query to its partner $\pi_{P'}^{j}$. Later (in the d-th or a later stage), π_P^i accepted a reset packet but $\mathcal{A}$ made no $\mathsf{Pack}(\pi_{P'}^{j}, \cdot, \mathtt{rst})$ queries.

The above captures the attacks in which the adversary forges a valid reset packet without breaking any Channel Security conditions. Note that such attacks are *undetectable* by the accepting party, as opposed to a network attacker that simply drops packets.

REMARK. Note that the payload integrity in secure channels is captured by Channel Security. Our msACCE-std and msACCE-pauth models *completely* capture the authentication (or integrity) of all packet fields in the transport and application layers. Furthermore, msACCE-pauth captures (network-layer) IP-Spoofing Prevention against weaker off-path attackers (i.e., those can only inject packets without observing the communication), but leaves other integrity attacks on low layers (e.g., network, link, and physical layers) uncovered. Such attacks may affect packet forwarding, node-to-node data transfer, or raw data transmission, which are outside the scope of our work.

4.4 Provable Security Analysis

Equipped with msACCE security models, we now analyze and compare the security of TFO+TLS 1.3, UDP+QUIC, and UDP+QUIC[TLS]. The security results are summarized in Table 1.2. As mentioned in the Introduction, by [11] results, no protocol achieves forward secrecy for 0-RTT keys or protects against 0-RTT data replays (which contribute to the first two rows in the table). We now move to the detailed analyses and start with TFO+TLS 1.3.

4.4.1 TLS 1.3 over TFO

Protocol Description

Referring to the msACCE protocol syntax, a TFO+TLS 1.3 2-RTT full handshake (see Fig. 4.3 left) is a 2-stage msACCE protocol in the full mode and a 0-RTT resumption handshake (see Fig. 4.3 right) is a 3-stage msACCE protocol in the resumption mode. Note that we focus only on the main components of the handshakes and omit more advanced features such as 0.5-RTT data, client authentication, and post-handshake messages (except `NewSessionTicket`). In a full handshake, the initial keys are set after sending or receiving `ServerHello` and the final keys (i.e., session keys) are set after sending or receiving `ClientFinished` (but only handshake messages up to `ServerFinished` are used for final key generation). In a 0-RTT resumption handshake, the parties set 0-RTT keys to encrypt or decrypt 0-RTT data, after sending or receiving `ClientHello`.

According to the TFO and TLS 1.3 specifications [26, 5], the TFO+TLS 1.3 header contains the TCP header (see Fig. 4.1). We ignore some uninteresting header fields such as port numbers and the checksum because modifying them only leads to redirected or dropped packets. Such adversarial capabilities are already considered in the msACCE security models. We thus define the header space $\mathcal{H}$ as containing the following fields: a 32-bit sequence number $\mathtt{sqn}$, a 32-bit acknowledgment number $\mathtt{ack}$, a 4-bit data offset $\mathtt{off}$, a 6-bit reserved field $\mathtt{resvd}$, a 6-bit control bits field $\mathtt{ctrl}$, a 16-bit window $\mathtt{window}$, a 16-bit urgent pointer $\mathtt{urgp}$, a variable-length ($\leq$ 320-bit) padded options $\mathtt{opt}$. For encrypted packets, $\mathcal{H}$ additionally contains the TLS 1.3 record header fields: an 8-bit type $\mathtt{type}$, a 16-bit version $\mathtt{ver}$, and a 16-bit length $\mathtt{len}$. We further define reset packets as those with the RST bit (i.e., the 4-th bit of $\mathtt{ctrl}$) set to 1. Note that $\mathsf{scfg\_gen}$ is undefined.

TLS 1.3 enforces different content types for encrypted key exchange and secure channel messages. For simplicity, we define $\mathcal{M}_{\mathsf{KE}}$ and $\mathcal{M}_{\mathsf{SC}}$ as consisting of bit strings differing in their first bits. $\mathcal{M}_{\mathsf{pRST}} = \varnothing$. In Appendix B, we define TLS 1.3's stateful AEAD scheme $\mathsf{sAEAD}_{\mathsf{TLS}} = (\mathsf{sGen}, \mathsf{sEnc}, \mathsf{sDec})$ based on the underlying nonce-based AEAD scheme $\mathsf{AEAD} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ (instantiated with AES-GCM [102] or others as documented in [5]).

We refer to Appendix A.1 for the remaining details of TFO and refer to [11, 15] for the detailed descriptions of TLS 1.3 handshake messages and key generations in earlier TLS

1.3 drafts as well as [5] for the latest updates.

Security Results

TFO+TLS 1.3's session identifier $\mathtt{sid}_{\mathrm{TLS}}$ is defined as all key exchange messages from `ClientHello` to `ServerFinished`, excluding TCP headers and IP addresses. The msACCE-std security of TFO+TLS 1.3 is by definition independent of TCP headers and is hence provided by the TLS 1.3 component. Previous works [103, 11, 10] only proved TLS 1.3's authenticated key exchange security, i.e., the stage keys are authenticated and indistinguishable from random ones under reasonable computational assumptions. In Appendix C.1, we show one can adapt their security results to prove TLS 1.3's Server Authentication and level-4 Channel Security in our msACCE-std model, by additionally relying on the level-4 AEAD security of $\mathsf{sAEAD}_{\mathrm{TLS}}$ (which can be reduced to the nonce-based AEAD security of the underlying AEAD as shown in [12]).

The msACCE-pauth security analyses are shown as follows.

IP-Spoofing Prevention. This security of TFO+TLS 1.3 is provided by the TFO component through TCP sequence number randomization and TFO cookies. By modeling the cookie generation function, an AES-128 block cipher, as a PRF $F : \{0,1\}^\lambda \times \{0,1\}^n \rightarrow \{0,1\}^n$, we have the following theorem with the proof in Appendix A.1:

Theorem 6. *For any PPT adversary $\mathcal{A}$ making at most q* Send *queries, there exists a PPT adversary $\mathcal{B}$ such that:*

$$\mathbf{Adv}^{\mathsf{ipsp}}_{\textit{TFO+TLS 1.3}}(\mathcal{A}) \leq |\mathcal{S}|\mathbf{Adv}^{\mathsf{prf}}_{F}(\mathcal{B}) + \frac{q}{\min\{2^{|\mathsf{sqn}|}, 2^n\}} .$$

KE Header Integrity. TFO+TLS 1.3 does not achieve this security notion because TCP headers are never authenticated. We find a new practical attack below, where a PPT adversary $\mathcal{A}$ can always get $\mathbf{Adv}^{\mathsf{int\text{-}keh}}_{\mathrm{TFO+TLS\ 1.3}}(\mathcal{A}) = 1$:

TFO Cookie Removal. $\mathcal{A}$ can first make $\pi_C^{i'}$ complete a full handshake with $\pi_S^{j'}$ (via

$\mathsf{Connect}, \mathsf{Send}$ queries), then query $\mathsf{Resume}(\pi_C^i, S, i')$ $(i' < i)$ to get the output packet $(\mathrm{IP}_C, \mathrm{IP}_S, H, pd)$, which is a SYN packet with a TFO cookie. $\mathcal{A}$ then modifies the `opt` field of H to get a new $H' \neq H$ that contains no cookie. The resulting SYN packet will be accepted by a new server oracle π_S^j, which will then respond with a SYN-ACK packet that does not contain a TFO cookie, indicating a fallback to the standard 3-way TCP. As a result, a 1-RTT handshake is needed to complete the connection and any 0-RTT data sent with SYN would be retransmitted. This eliminates the entire benefit of TFO without being detected, resulting in reduced performance and increased handshake latency. A similar attack is possible by removing the TFO cookie in a server's SYN-ACK packet.

Interestingly, clients are supposed to cache negative TFO responses and avoid sending TFO connections again for a lengthy period of time. This is because the most likely explanation for this behavior is that the server does not support TFO, but only standard TCP [26]. As a result, performing this attack for a single connection prevents TFO from being used with this server for a lengthy time period (i.e., days or weeks).

KE Payload Integrity. TFO+TLS 1.3 is secure in this regard simply because $\mathtt{sid}_{\mathrm{TLS}}$ consists of the payloads of all key exchange packets exchanged between the communicating parties before the client set its session key. That is, for any client oracle that has a matching conversation with any server oracle, by definition they observe the same $\mathtt{sid}_{\mathrm{TLS}}$ and hence no key exchange packet payload can be modified, i.e., $\mathbf{Adv}^{\mathsf{int\text{-}kep}}_{\mathrm{TFO+TLS\ 1.3}}(\mathcal{A}) = 0$ for any PPT adversary $\mathcal{A}$.

SC Header Integrity. TFO+TLS 1.3 does not achieve this security notion again because of the unauthenticated TCP headers. A PPT adversary $\mathcal{A}$ can get $\mathbf{Adv}^{\mathsf{int\text{-}h}}_{\mathrm{TFO+TLS\ 1.3}}(\mathcal{A}) = 1$ by either modifying the TCP header of an encrypted packet (e.g., reducing the `window` value) or by forging a header-only packet (e.g., removing the payload of an encrypted packet and changing its `ack` value). Such packets are valid and will be accepted by the receiving session oracle.

The above fact exposes the adversary's ability to arbitrarily modify or even entirely forge the information in the TCP header, which is being relied on to provide reliable delivery, in-order delivery, flow control, and congestion control for the targeted flow. This leads to a whole host of availability attacks that the networking community has been slowly uncovering via manual investigation over the last 30 years [82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95]. Some of the practical attacks are described as follows.

TCP Flow Control Manipulation. An adversary with access to the communication channel can impact TCP's flow control mechanism to decrease the sending rate or stall the connection by modifying TCP's `window` header field. This field controls the amount of received data the sender of this packet is prepared to buffer. By reducing this quantity, the throughput of the connection can be reduced and if it is set to zero the connection will completely stall.

One example of this attack would be to modify the window field to zero in a TCP packet containing a TLS-encrypted HTTP request. Since TCP headers are not authenticated, this modification will not be detected. As a result, when the server receives this request and attempts to send the response, it will believe that the client cannot currently accept any data and will delay sending the response. After some timeout, TCP will probe the client with a single packet of data to determine whether the window is still zero. If the adversary also modifies the responses to these probes, the connection will remain stalled indefinitely; otherwise, the connection will eventually recover after a lengthy delay.

TCP Acknowledgment Injection. An adversary who can observe a target connection and forge packets can inject new acknowledgment packets into the TCP connection. Acknowledgment packets have no data making them undetectable by either TLS or the application. However, they are used by congestion control to determine the allowed sending rate of a connection.

Injecting duplicate or very slowly increasing acknowledgments can be used to slow a target connection down drastically. [95] demonstrated a 12x reduction in throughput using this approach with the attacker required to expend only 40Kbps. This, of course, represents a significant performance degradation for a TFO+TLS 1.3 connection.

Injecting acknowledgments can also be used to dramatically increase the sending rate of a connection, turning it into a firehose that an attacker can point at their desired target. This is done by sending acknowledgments for data that has not been received yet, an attack known as Optimistic Ack [82]. This attack renders TCP insensitive to congestion and can completely starve competing flows. It could be used with great effect to cause denial of service against a server or the Internet infrastructure as a whole [104].

Reset Authentication. TFO+TLS 1.3 is insecure in this sense because its reset packet, TCP Reset, is an unauthenticated header-only packet. This leads to a practical attack below, where a PPT adversary $\mathcal{A}$ always gets $\mathbf{Adv}^{\mathsf{rst\text{-}auth}}_{\text{TFO+TLS 1.3}}(\mathcal{A}) = 1$:

TCP Reset Attack. $\mathcal{A}$ can first make two session oracles complete a handshake using $\mathsf{Connect}, \mathsf{Send}$ queries, then use $\mathsf{Pack}, \mathsf{Deliver}$ queries to let them exchange secure channel packets. By observing these packet headers, $\mathcal{A}$ can easily forge a valid reset packet by setting its RST bit to 1 and the remaining header fields to reasonable values. This attack will cause TCP to tear down the connection immediately without waiting for all data to be delivered.

Note that even an off-path adversary who can only inject packets into the communication channel may be able to accomplish this attack. The injected TCP reset packet needs to be within the receive window for the client or server, but [94] demonstrated that a surprisingly small number of packets is needed to achieve this, thanks to the large receive windows typically used by implementations.

4.4.2 QUIC over UDP

Protocol Description

Referring to the msACCE protocol syntax, an UDP+QUIC 1-RTT full handshake (see Fig. 4.4 left) is a 2-stage msACCE protocol in the full mode and a 0-RTT resumption handshake (see Fig. 4.4 right) is a 2-stage msACCE protocol in the resumption mode. The initial keys are set after sending or receiving `ClientHello` and the final keys (i.e., session keys) are set after sending or receiving `ServerHello`.

According to the UDP and QUIC specifications [24, 97, 105], the UDP+QUIC header contains the UDP header (see Fig. 4.2) and the QUIC header (described below). As with the TCP header, we ignore the port numbers and checksum in the UDP header. Similarly, we also ignore the UDP length field because it only affects the length of the QUIC header and payload. We thus can completely omit the UDP header and define the header space $\mathcal{H}$ as containing the following fields: an 8-bit public flag `flag`, a 64-bit connection ID `cid`, a variable-length (≤ 48 bits) sequence number `sqn`, and other optional fields. We further define reset packets as those with the PUBLIC_FLAG_RESET bit (i.e., the 7-th bit of `flag`) set to 1. A reset packet header only contains `flag` and `cid`.

As with TLS 1.3, for UDP+QUIC we define $\mathcal{M}_{\mathsf{KE}}$ and $\mathcal{M}_{\mathsf{SC}}$ as consisting of bit strings differing in their first bits. $\mathcal{M}_{\mathsf{pRST}} = \varnothing$. In Appendix B, we define QUIC's stateful AEAD scheme $\mathsf{sAEAD}_{\text{QUIC}} = (\mathsf{sGen}, \mathsf{sEnc}, \mathsf{sDec})$ based on the underlying nonce-based AEAD scheme $\mathsf{AEAD} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ (instantiated with AES-GCM [102]).

We refer to [32] for the detailed descriptions of $\mathsf{scfg\_gen}$ and QUIC handshake messages and key generations.

Security Results

UDP+QUIC's session identifier $\mathtt{sid}_{\text{QUIC}}$ is defined as the `ClientHello` payload and `ServerHello`, excluding IP addresses. The msACCE-std security of UDP+QUIC follows from prior works as we discuss in Appendix C.2. Note that UDP+QUIC only achieves level-1 Channel Security, but, as discussed in [32], QUIC implicitly prevents packet reordering by authenticating `sqn` in the packet header. It also prevents replays and dropping with frame sequence numbers encrypted in the payload. Therefore, UDP+QUIC essentially achieves level-4 authentication as TLS 1.3 does.

The msACCE-pauth security analyses are shown as follows.

IP-Spoofing Prevention. In [32], QUIC has been proven secure against IP spoofing based on the AEAD security. Their IP-spoofing security notion is the same as our IP-Spoofing Prevention notion for UDP+QUIC except that ours additionally captures attacks in full sessions. However, since source-address tokens are validated in both full and resumption sessions, their results can be trivially adapted to show that UDP+QUIC achieves IP-Spoofing Prevention.

KE Header and Payload Integrity. UDP+QUIC does not achieve these security notions because its first-round key exchange messages, i.e., `InchoateClientHello` and `ServerReject`, and any invalid `ClientHello` are not fully authenticated. Interestingly, a variety of existing attacks on QUIC's availability discovered in [32] are all examples of key exchange packet manipulations (e.g., the server config replay attack, connection ID manipulation attack, etc.), but these attacks cause connection failure and hence are easy to detect. However, successful attacks breaking KE Header or Payload Integrity will be harder (if not impossible) to detect.

For KE Header Integrity, we do not find any harmful attacks but theoretical attacks exist. For instance, a PPT adversary $\mathcal{A}$ can get $\mathbf{Adv}^{\mathsf{int\text{-}keh}}_{\text{UDP+QUIC}}(\mathcal{A}) = 1$ as follows. $\mathcal{A}$ can first query $\mathsf{Connect}(\pi^i_C, S)$ to get the output packet $(\text{IP}_C, \text{IP}_S, H, pd)$, then modify the $\mathtt{flag}$ and $\mathtt{sqn}$ fields of H to get a new header $H' \neq H$ that only changes $\mathtt{sqn}$'s length but not its value. The resulting packet will be accepted by a new server oracle π^j_S. This attack has no practical impact on UDP+QUIC but it successfully modifies the protocol header without being detected.

For KE Payload Integrity, we find a new practical attack described below where a PPT adversary $\mathcal{A}$ can get $\mathbf{Adv}^{\mathsf{int\text{-}kep}}_{\text{UDP+QUIC}}(\mathcal{A}) \approx 1$:

ServerReject Triggering. $\mathcal{A}$ can first let $\pi^{i'}_C$ complete a full handshake with $\pi^{j'}_S$ with $\mathsf{Connect}, \mathsf{Send}$ queries, then query $\mathsf{Resume}(\pi^i_C, S, i')$ $(i' < i)$ to get the output `ClientHello` packet. $\mathcal{A}$ then modifies its payload by replacing the source-address token `stk` with a random value, which with high probability is invalid. Sending this modified packet to a new server oracle π_S^j will trigger a `ServerReject` packet containing a new valid `stk`. This as a result downgrades the original 0-RTT resumption connection to a full 1-RTT connection, which causes increased latency and results in the retransmission of any 0-RTT data. Note that this attack is hard to detect because π_C^i may think its original $\mathtt{stk}'$ has expired (although this does not happen frequently).

SC Header Integrity. UDP+QUIC is secure in this regard because it does not allow header-only packets to be sent in the secure channel phases and the *entire* protocol header is taken as the associated data authenticated by the underlying encryption scheme. Therefore, UDP+QUIC's SC Header Integrity can be reduced to its level-1 Channel Security. Formally, for any PPT adversary $\mathcal{A}$ there exists a PPT adversary B such that $\mathbf{Adv}_{\text{UDP+QUIC}}^{\textsf{int-h}}(\mathcal{A}) \leq \mathbf{Adv}_{\text{UDP+QUIC}}^{\textsf{cs-1}}(B)$.

Reset Authentication. UDP+QUIC does not achieve this security notion because, similar to TCP Reset, its reset packet `PublicReset` is not authenticated either. In the following availability attack, a PPT adversary $\mathcal{A}$ can always get $\mathbf{Adv}_{\text{UDP+QUIC}}^{\textsf{rst-auth}}(\mathcal{A}) = 1$:

PublicReset Attack. $\mathcal{A}$ can first make two session oracles complete a handshake using Connect, Send queries, then use Pack, Deliver queries to let them exchange secure channel packets. By observing these packet headers, $\mathcal{A}$ can easily forge a valid (plaintext) reset packet by setting its PUBLIC_FLAG_RESET bit to 1 and the remaining packet fields to reasonable values (which is easy because it simply contains the connection ID `cid`, the sequence number of the rejected packet, and a nonce to prevent replay). This attack will cause similar effects as described in the TCP Reset attack. Note that this vulnerability is fixed in QUIC[TLS] shown below.

4.4.3 QUIC[TLS] over UDP

Protocol Description

As mentioned in the Background, QUIC[TLS] replaces QUIC's key exchange with the TLS 1.3 key exchange. So, as with TLS 1.3, a UDP+QUIC[TLS] 2-RTT full handshake is a 2-stage msACCE protocol in the full mode and a 0-RTT resumption handshake is a 3-stage msACCE protocol in the resumption mode. The stage keys are set in the same way as in TLS 1.3.

The header fields (as specified in [27]) are similar to those in UDP+QUIC. Reset packets are defined as those whose first two header bits are 01. $\mathsf{scfg\_gen}$ is undefined. UDP+QUIC[TLS] also enforces different frame types for encrypted key exchange, secure channel, and pre-reset messages. For simplicity, we define $\mathcal{M}_{\mathsf{KE}}, \mathcal{M}_{\mathsf{SC}}, \mathcal{M}_{\mathsf{pRST}}$ as consisting of bit strings differing in their first two bits. UDP+QUIC[TLS]'s stateful encryption scheme is the same as $\mathsf{sAEAD}_{\mathrm{QUIC}}$ based on the underlying nonce-based AEAD scheme $\mathsf{AEAD} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ (instantiated with AES-GCM [102] or others as documented in [5]).

QUIC[TLS] still provides source validation with a secure token generated by the server, similar to the case in Google's QUIC. We discuss QUIC[TLS]'s stateless reset mechanism later in the security analysis of Reset Authentication and refer to [27, 28] for the detailed UDP+QUIC[TLS] handshake messages and key generations.

Security Results

UDP+QUIC[TLS]'s session identifier $\texttt{sid}_{\mathrm{QUIC[TLS]}}$ is defined as $\texttt{sid}_{\mathrm{TLS}}$. By construction, UDP+QUIC[TLS] inherits the msACCE-std security from TLS 1.3 (but using QUIC's underlying encryption scheme). That is, it achieves level-1 Channel Security and implicitly achieves level-4 authentication as discussed before. UDP+QUIC[TLS] has a similar source-validation token scheme as QUIC. If the token is generated with an authenticated encryption scheme, the IP-Spoofing Prevention security of UDP+QUIC[TLS] can be reduced to the encryption scheme's authenticity security. However, such a source-validation scheme suffers from an availability attack against KE Payload Integrity similar to ServerReject Triggering for UDP+QUIC, where the adversary replaces the source-validation token with a random value to downgrade a 0-RTT resumption connection. As noted in [28], an adversary can also modify the unauthenticated ACK frames in the Initial packets without being detected. Furthermore, UDP+QUIC[TLS] achieves SC Header Integrity in the same way as UDP+QUIC. We are only left to show its security of KE Header Integrity and Reset Authentication.

KE Header Integrity. UDP+QUIC[TLS] does not achieve these security notions because its first-round Initial packets (see [27]) are not fully authenticated. For instance, a PPT adversary $\mathcal{A}$ can get $\mathbf{Adv}^{\mathsf{int\text{-}keh}}_{\text{UDP+QUIC[TLS]}}(\mathcal{A}) = 1$ as follows. $\mathcal{A}$ first queries $\mathsf{Connect}(\pi^i_C, S)$ to get π^i_C's Initial packet $(\text{IP}_C, \text{IP}_S, H, pd)$. Then, as described in [28], $\mathcal{A}$ can decrypt this packet with its Destination Connection ID `dcid` in H, change it to another value `dcid`$'$, and re-encrypt the whole packet with this new `dcid`$'$. The resulting packet $(\text{IP}_C, \text{IP}_S, H', pd')$, where $H \neq H'$, is valid and will be accepted by a new server oracle π^j_S without being detected by the client. However, this is only a theoretical attack with no practical impact.

Reset Authentication. In UDP+QUIC[TLS], the stateless reset works as follows. One party generates a 128-bit reset token using its static key and a random 64-bit `cid` as input. Then this token (carried within the pre-reset message) is sent to the other party in a secure channel phase. Later, the same party that generated this token can perform a stateless reset by regenerating the token and sending it to the other party in clear (via a reset packet).

The Reset Authentication security of UDP+QUIC[TLS] can be reduced to its level-1 Channel Security and the PRF security of the reset token generation function $F : \{0,1\}^\lambda \times \{0,1\}^{|\mathtt{cid}|} \rightarrow \{0,1\}^n$ as shown in the theorem below with the proof in Appendix A.2:

Theorem 7. *For any PPT adversary $\mathcal{A}$ delivering at most q_{rst} forged reset packets (via*

Deliver *queries), there exist PPT adversaries $\mathcal{B}$ and $\mathcal{C}$ such that:*

$$\mathbf{Adv}^{\textsf{rst-auth}}_{\text{UDP+QUIC[TLS]}}(\mathcal{A}) \quad \leq \quad |\mathcal{P}|\mathbf{Adv}^{\textsf{prf}}_{F}(\mathcal{B}) + \mathbf{Adv}^{\textsf{cs-1}}_{\text{UDP+QUIC[TLS]}}(\mathcal{C}) + \frac{|\mathcal{P}|N^2}{2^{|\texttt{cid}|}} + \frac{q_{\textsf{rst}}}{2^n}.$$

CHAPTER 5
PROVABLE SECURITY ANALYSIS OF FIDO2

5.1 Introduction

As motivated in Subsection 1.2.3, our goal in this work is to perform the first provable security analysis of the latest FIDO2 protocols.

5.1.1 Related Work and Focus

There are not much security analysis about FIDO protocols. To the best of our knowledge, the only existing security results are as follows. Hu and Zhang [106] analyzed the security of FIDO UAF 1.0 and identifies several vulnerabilities in different attack scenarios. Later, Panos *et al.* [107] analyzed FIDO UAF 1.1 and explored some potential attack vectors and vulnerabilities. However, both works were informal. FIDO U2F and WebAuthn were analyzed with formal methods using the applied pi-calculus and ProVerif tool [108, 109, 110]. Regarding an older version of FIDO U2F, Pereira *et al.* [108] presented a server-in-the-middle attack and Jacomme and Kremer [109] further analyzed it with a structured and fine-grained threat model for malwares. Guirat and Halpin [110] confirmed the authentication security provided by WebAuthn while pointed out that the claimed privacy properties (i.e., account unlinkability) failed to hold due to the same attestation key used among different servers.

However, there is *no* provable security analysis of the new FIDO2 protocols despite their fast deployment process. In particular, CTAP2 has not been analyzed yet and the formal methods results for WebAuthn [109] are quite insufficient. As noted by the authors, their model "makes a number of simplifications and so much work is needed to formally model the complete protocol as given in the W3C specification". Moreover, security proofs generated by formal methods are associated with the *symbolic* model, which is too abstract to capture subtle attacks. In particular, a symbolic model treats the underlying cryptographic building blocks as “ideal” and hence does not give us guidance of what practical known security properties the cryptographic primitives should satisfy. For instance, if a protocol uses a digital signature scheme, the formal methods will not indicate whether one should employ a signature scheme that is unforgeable for a given message, existentially unforgeable under chosen message attack, or strongly unforgeable under chosen message attack.

In comparison, the provable security approach used to prove security of cryptographic protocols utilizes a *computational* model, which is much less abstract and is tied to the practical adversarial capabilities much more closely. The proofs proceed by reduction, i.e., showing that if an attacker breaks the security of the target protocol then the security or computational hardness assumption about some building block (e.g., strong unforgeability of the signature scheme) does not hold. Proofs in the computational model provide stronger security guarantees than the formal methods proofs, but they are often more complicated.

Our goal is to provide the first provable security analysis of the latest FIDO2 protocols to help practitioners understand their security guarantees and vulnerabilities. More precisely, we focus on the main FIDO2 components WebAuthn [37] and CTAP2 [38].

5.1.2 Our Contributions

We perform the first thorough cryptographic analysis of FIDO2 protocols using the provable security approach: first defining the protocol syntax, then designing its security model that specifies the adversarial capabilities and formal security goals, and finally proving security by reduction to the security of the building blocks or identifying attacks based on the model. To better understand and improve each component protocol, our analysis is conducted in a *modular* way. That is, we analyze CTAP2 and WebAuthn separately and then derive the overall security of FIDO2 by composing their security.

We start our analysis with the more complex CTAP2 protocol and define a *PIN-based Authenticator Setup and Authenticated Channel Establishment (PASACE)* protocol to formalize the general syntax of the core CTAP2 components. Although CTAP2 by its name may suggest a two-party protocol, our PASACE protocol actually involves the user as an additional party and captures human interactions with the client and authenticator (e.g., the user typing its PIN into the client or rebooting the authenticator). PASACE runs in three phases as follows. First, in the authenticator setup phase, the user "embeds" its PIN into the authenticator via the client and as a result the authenticator stores a PIN-related state that we call a transformed PIN. Then, the client with the same input PIN is "bound" to the set-up authenticator in the binding phase and each party ends up with setting a (perhaps different) binding state. Finally, in the authenticated channel phase, the client is able to send any authenticated message (prepared with its binding state) to the authenticator, which accepts or rejects the received message after verifying it with the binding state. Note that the final established authenticated channel is *unidirectional*, i.e., it only guarantees authenticated access from the client to the authenticator but not the other way.

As mentioned before, the security goal of CTAP2 is essentially to grant a client (or multiple clients) exclusive access to the user's authenticator. To achieve this, the authenticator privately sends its associated secret called pinToken (generated upon power-up) to the trusted client, where the pinToken will be used as the binding state to authenticate and verify messages sent to the authenticator. By the CTAP2 design, each authenticator is associated with a *single* pinToken, so multiple clients establish multiple authenticated channels with the same authenticator using the *same* pinToken. This limits the security of CTAP2 authenticated channels to a weak sense: in order for a particular channel between an authenticator and a client to be secure (i.e., no attacker can forge authenticated messages in that channel), *no* clients bound to the same authenticator are allowed to be compromised.

In our security model for PASACE protocols, we first define the notion Unforgeability (UF) to cover the above weak security and then define Strong Unforgeability (SUF) to capture *strong* fine-grained security. Unlike UF, SUF allows the attacker to compromise any clients except for the client in the target channel. As explained in Section 5.3, SUF also covers certain *forward secrecy* guarantees for authentication. For both notions, we consider a powerful attacker that can manipulate the communication between parties, compromise clients (that are not bound to the target authenticator) to reveal the binding states, and corrupt users (that did not set up the target authenticator) to learn their secret PINs.

However, we assume the authenticator setup phase is executed in a *trusted* way, i.e., its execution is protected from malicious manipulation (but eavesdropping is allowed), because otherwise an attacker can "embed" any PIN into the authenticator and no security is likely to hold. On the other hand, the binding phase may be executed in either a trusted or an unprotected public environment. To enable comprehensive security analysis, our model captures both cases for UF and SUF, resulting in two more weaker notions denoted by UF-t and SUF-t respectively for the trusted binding case. Unlike a trusted setup, in the trusted binding setting, an attacker can still interact with the authenticator and the client during the binding phase, except that it cannot interfere with a honest binding execution. As a summary of our four notions, by definition SUF is the strongest security notion and UF-t is the weakest one, but UF and SUF-t are *incomparable* as indicated by our separation result discussed in Section 5.5.

Based on our security model, we first prove that a simplified (but as secure) version of CTAP2 only achieves the weakest UF-t security. CTAP2 cannot achieve UF security because it uses unauthenticated Diffie-Hellman key exchange in the binding phase, which is vulnerable to man-in-the-middle (MITM) attacks. To achieve stronger security, we propose the *PIN-based Authenticator Setup and Key Exchange (PASKE)* protocol, which utilizes a *Password-Authenticated Key Exchange (PAKE)* protocol. PAKE takes as input a common password and outputs the same random session key for both party. The key observation is that the transformed PIN shared between the authenticator and the client (which gets the user PIN as input) can be treated as a common password to run PAKE. The resulting session key is used as the binding state to build the authenticated channel. Thanks to the *independent* random keys generated by PAKE, we prove that PASKE actually achieves the strongest SUF security. Furthermore, by comparing its efficiency with CTAP2, we argue that PASKE comes with only very little overhead.

Next, we define an *Authenticator-assisted Passwordless User Authentication (APlUA)* protocol to capture the WebAuthn protocol syntax. Our APlUA protocol considers all four parties involved in WebAuthn: a user, an authenticator, a client, and a server (often referred to as a relying party). Similar to PASACE, an APlUA protocol covers all types of communications between parties including human interactions with the client and authenticator.

In our security model for APlUA protocols, we first define the basic security notion User Authentication (UA), which is however more complicated than one might expect due to the involvement of multiple parties. Roughly, UA guarantees that if a server accepts the authentication from a user then not only the registered authenticator must be used to authenticate to the server (which is the only authentication property considered in [110]) but also the following holds: 1) the authentication must be approved on the registered authenticator via the human communication channel (e.g., by pressing a button); 2) the authentication must go through a trusted client (i.e., a client that has access to an authenticated channel towards the authenticator); and 3) the server that accepts must be the user's intended server. Note that the second condition helps forbid malicious user authentication even if the authenticator is stolen by the attacker because it still needs to compromise or steal a trusted client.

Though strong as it seems, UA does not guarantee any security for a user if *any* of the trusted clients (and hence the authenticated channels) associated with the registered authenticator is compromised. We hence define a strong security notion Strong User Authentication (SUA) to achieve authentication security even if some of the channels are compromised.

We prove that WebAuthn is UA secure when the underlying client-to-authenticator authenticated channels achieve at least weak security and other building blocks are secure, i.e., the hash function is collision-resistant and the signature scheme is unforgeable. Then, we show that WebAuthn does not achieve SUA security even if its underlying channels are strongly secure, which contradicts our intuition. The reason behind this is that in WebAuthn, though an authenticator can distinguish between different strong channels, a user cannot detect which channel is used. As a result, an attacker can trick a user to approve a malicious authentication request to a different server (that registered the same authenticator as the intended server) through a compromised channel, while the user might think the authentication goes through the specified uncompromised channel. To fix this issue, we propose an augmented WebAuthn protocol called WebAuthn+ and prove that it achieves SUA security. In WebAuthn+, the user specifies a recognizable unique name for each authenticated channel and inputs it to the client, which then sends the name to the user's authenticator through the authenticated channel. Later, upon each request for user approval the authenticator shows him the channel name. This augmented mechanism may require a display on the authenticator but enables the user to distinguish between different channels and hence achieve stronger authentication security.

Finally, we present our composition theorem that shows that a PASACE protocol can be securely composed with an APlUA protocol. With this composition theorem, we derive that the overall FIDO2 achieves UF-t and UA security, while our new construction that integrates PASKE and WebAuthn+ achieves the strongest SUF and SUA security.

To summarize, we provide the first thorough provable security analysis of the FIDO2 protocols: CTAP2 and WebAuthn. We hope our models and provable security results will help clarify the security guarantees of the FIDO2 protocols and expect our proposed constructions to facilitate the design and deployment of more secure passwordless user authentication protocols.

5.2 IND-1\$PA Security for Deterministic Encryption

Before defining our security models, we first introduce a new security notion used in our analysis that we call *indistinguishability under one-time chosen and then random plaintext attack (IND-1\$PA).*

Definition 3 (IND-1\$PA). *Consider a deterministic encryption scheme* $\mathsf{DE} = (\mathcal{K}, \mathsf{Enc}, \mathsf{Dec})$ *and the following security experiment associated with an adversary* $\mathcal{A}$*. In the beginning, sample a random key* $k \xleftarrow{\$} \mathcal{K}$ *and a random bit* $b \xleftarrow{\$} \{0,1\}$*. Then,* $\mathcal{A}$ *is granted access to the following encryption oracles:*

- $\mathcal{O}_{\mathsf{LR}}(m_0, m_1)$*: returns* $\mathsf{Enc}(k, m_b)$*, where the input* m_0, m_1 *are of the same bit length which is a multiple of the block size* `bl`*. This oracle is queried only once (if any) and has to be the first query.*

- $\mathcal{O}_{\$\mathsf{LR}}(l, \texttt{same}$ *or* $\texttt{rand})$*: if the second input is* `same`*, samples a random message* $m \xleftarrow{\$} \{0,1\}^{l \cdot \texttt{bl}}$ *and returns* $(m, \mathsf{Enc}(k, m))$*; if the second input is* `rand`*, samples two independent random messages* $m_0 \xleftarrow{\$} \{0,1\}^{l \cdot \texttt{bl}}$*,* $m_1 \xleftarrow{\$} \{0,1\}^{l \cdot \texttt{bl}}$ *and returns* $(m_0, m_1, \mathsf{Enc}(k, m_b))$*.*

In the end, $\mathcal{A}$ *outputs a bit* b' *as its guess of* b*. Its advantage measure* $\mathbf{Adv}^{\mathsf{ind\text{-}1\$pa}}_{\mathsf{DE}}(\mathcal{A})$ *is defined as* $|2\Pr(b = b') - 1|$*, which is equivalent to* $|\Pr(\mathcal{A} \Rightarrow 1 \mid b = 0) - \Pr(\mathcal{A} \Rightarrow 1 \mid b = 1)|$.

We say DE *is IND-1\$PA secure if for any probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$*,* $\mathbf{Adv}^{\mathsf{ind\text{-}1\$pa}}_{\mathsf{DE}}(\mathcal{A}) = \mathsf{negl}(\log|\mathcal{K}|)$.

Comparison to other IND-CPA Security Notions. Note that our IND-1\$PA security is implied by the classical (deterministic) IND-CPA security (where $\mathcal{O}_{\mathsf{LR}}$ can be queried multiple times as long as messages in each world never repeat) because the random LR oracle $\mathcal{O}_{\$\mathsf{LR}}$ can be simulated by $\mathcal{O}_{\mathsf{LR}}$. However, IND-1\$PA does not imply IND-CPA. In particular, the encryption scheme CBC_0 (described later) used by CTAP2 is IND-1\$PA secure (see Theorem 8), but it is obviously not IND-CPA secure for long messages (e.g., of length more than one block size). On the other hand, IND-1\$PA clearly implies *one-time* IND-CPA for which no $\mathcal{O}_{\$\mathsf{LR}}$ queries are allowed after the single $\mathcal{O}_{\mathsf{LR}}$ query. Such one-time IND-CPA security was defined as security against passive attacks by Cramer and Shoup [111]. Note that the other direction is not true, e.g., the one-time pad is one-time IND-CPA secure but not IND-1\$PA secure.

5.3 PIN-Based Authenticator Setup and Authenticated Channel Establishment

In this section, we define the syntax and security model for *PIN-based Authenticator Setup and Authenticated Channel Establishment (PASACE)* protocols.

5.3.1 Protocol Syntax

A PASACE protocol is an interactive protocol among a human user, an authenticator and a client. First, the user that has a PIN sets up the authenticator via the client, i.e., "embedding" the user PIN in the authenticator. Then, they establish a client-to-authenticator channel to allow the client sending authenticated messages (along with unauthenticated auxiliary data) to the authenticator.

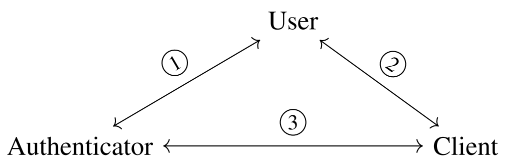


Figure 5.1: Communication Channels.

The possible communication channels are represented as double-headed arrows in Figure 5.1. Following the intuitive definitions of *human-compatible communications* by Boldyreva *et al.* [39], we also require messages sent to the user be *human-readable* and those sent by the user be *human-writable*.[1] The user PIN needs to be *human-memorizable*.

The protocol is associated with the security parameter $\lambda \in \mathbb{N}_+$, a PIN space $\mathcal{PIN} \subseteq \{0,1\}^*$, a transformed-PIN space $\mathcal{TPIN} \subseteq \{0,1\}^\lambda$, an authenticator power-up state space $\mathcal{PS} \subseteq \{0,1\}^*$, a binding state space $\mathcal{BS} \subseteq \{0,1\}^\lambda$, a message space $\mathcal{M} \subseteq \{0,1\}^* \setminus \{\varepsilon\}$, an auxiliary data space $\mathcal{X} \subseteq \{0,1\}^*$, a deterministic PIN transformation function trans that takes as input a PIN $pin \in \mathcal{PIN}$ and outputs a transformed PIN $tpin \in \mathcal{TPIN}$, and an authenticator power-up state generation function $\mathsf{ps\_gen}$ that takes as input 1^λ and outputs a power-up state $ps \in \mathcal{PS}$. All the above functions are efficient.

In the beginning of the protocol's execution, the authenticator takes as input its associated power-up state ps (generated by running $\mathsf{ps\_gen}(1^\lambda)$) and the user takes as input its long-term secret PIN $pin \in \mathcal{PIN}$. During its execution, each party (except the user) can keep states that are initialized to the empty string ε.

A PASACE protocol consists of the following phases where the user sends the first messages in the first two phases:

1) Authenticator Setup. This phase is optional and executed *at most once* for each authenticator. At the end of this phase, the authenticator sets its transformed PIN $\mathtt{tpin}$.

2) Binding. At the end of this phase, the authenticator and client each sets its (perhaps different) binding state $\mathtt{bs} \in \mathcal{BS}$ (if any[2]) on success or halts (with $\mathtt{bs}$ unset) on failure.

3) Authenticated Channel. In this phase, the client sends one or more authenticated messages along with auxiliary data to the authenticator.[3] For each $(M, x) \in \mathcal{M} \times \mathcal{X}$ to be sent, the client outputs $\widehat{M}$ and sends $(\widehat{M}, x)$ to the authenticator which either accepts or rejects the received message.

We say an authenticator *rejects* a received message if processing it results in an error (defined according to the protocol), and *accepts* it otherwise. During the protocol execu-

[1]For PASACE protocols, we regard understandable information from Internet browsing as human-readable and typing in a PIN or rebooting an authenticator as human-writable.

[2]The authenticator may have no separate binding state but resort to its power-up state.

[3]Note that the established authenticated channel is unidirectional, i.e., only messages sent from the client to the authenticator are authenticated but not the other direction. This is also why we call the second phase "binding" instead of "pairing" where the latter usually implies symmetry.

tion, the user may *reboot* (i.e., re-power-up) the authenticator.

Correctness. First, if an authenticator is rebooted, its power-up state is refreshed by running $\mathsf{ps\_gen}$ again and all of its binding states (if any) are erased. Then, consider any user (with input pin), authenticator, and client running a PASACE protocol without authenticator reboots. At the end of the authenticator setup phase, the authenticator sets its transformed PIN $\mathtt{tpin}$ equal to $\mathtt{trans}(pin)$. The binding phase succeeds if the authenticator's transformed PIN $\mathtt{tpin}$ was set equal to $\mathtt{trans}(pin)$ in the beginning of this phase. If the binding phase succeeded, then in the authenticated channel phase the authenticator accepts all messages sent by the client.

5.3.2 Security Model

We now formally define the security model for PASACE protocols.

Protocol Entities. The set of parties $\mathcal{P}$ consists of three disjoint type of parties: user $\mathcal{U}$, authenticators $\mathcal{T}$, and clients $\mathcal{C}$, i.e., $|\mathcal{P}| = |\mathcal{U}| + |\mathcal{T}| + |\mathcal{C}|$.

Session Oracles. To capture multiple sequential and parallel protocol executions, each party $P \in \mathcal{P}$ is associated with a set of session oracles $\pi_P^1, \pi_P^2, \ldots$, where π_P^i models P executing a protocol instance in session $i \in \mathbb{N}_+$.

Communication Channel Security. For an PASACE execution, we assume the following security properties for the communication channels shown in Figure 5.1:

- Human communications (①②) are authenticated and private.
- Client-authenticator communications (③) are not protected, i.e., neither authenticated nor private.

Security Experiments. In the beginning of the experiments, run $\{ps_T \xleftarrow{\$} \mathsf{ps\_gen}(1^\lambda)\}_{T\in\mathcal{T}}$ to generate power-up states for all authenticators, sample independent and random PINs $\{pin_U \xleftarrow{\$} \mathcal{PIN}\}_{U\in\mathcal{U}}$ for all users, and initialize global states (e.g., transformed PIN) of all parties and local states (e.g., binding state) of all session oracles. According to the syntax of PASACE protocols, each authenticator has only one transformed PIN, but each authenticator (resp. client) may have multiple binding states, one for each different "bound" client (resp. authenticator) oracle. Let $N \in \mathbb{N}_+$ denote the maximum number of PASACE protocol instances for each party. We assume the human communications (①② in Figure 5.1) are authenticated and private. The adversary $\mathcal{A}$ interacts with session oracles via the following queries:

• $\mathsf{Execute}(\pi_U^l, \pi_T^i, \pi_C^j)$, for $U \in \mathcal{U}, T \in \mathcal{T}, C \in \mathcal{C}, l, i, j \in [N]$.

This query asks $\pi_U^l, \pi_T^i, \pi_C^j$ to execute the protocol (honestly) for the first two phases (or only the second phase if T was already set up) then returns a transcript (excluding human communications) of its execution. If the authenticator setup phase is successfully executed in this query, we say T was *set up by* U.

This query allows the adversary to access honest protocol executions.

• $\mathsf{Connect}(\pi_U^l, \pi_T^i, \pi_C^j)$, for $U \in \mathcal{U}, T \in \mathcal{T}, C \in \mathcal{C}, l, i, j \in [N]$.

This query asks π_U^l, that intends to bind π_T^i and π_C^j, to send the first message to π_C^j (through an authenticated and private channel) in the binding phase. The output of π_C^j through channel ③ is returned to $\mathcal{A}$ instead of π_T^i.

This query allows the adversary to ask a specified user oracle to start a binding phase between a specified client oracle and a specified authenticator oracle, then get the message output by the client oracle. Note that we do not have a similar "connect" query in the authenticator setup phase because we assume that phase is always performed in a trusted way (often referred to as a *trusted setup* in practice), i.e., not subject to any kind of attacks. For the same reason, we do not have a "send" query in the authenticator setup phase either as shown below.

• $\mathsf{Send}(\pi_P^i, m, n)$, for $P \in \mathcal{P} \setminus \mathcal{U}, i \in [N], m \in \{0,1\}^* \cup \{\mathsf{Reboot}\}, n \in [3]$.

If $P \in \mathcal{T}$, $m = \mathsf{Reboot}$ and $n = 1$, this query reboots authenticator P and returns. If π_P^i is in the binding phase or the authenticated channel phase, this query sends m to a non-user oracle π_P^i through communication channel ⓝ shown in Figure 5.1 and returns its response (if any) through the same channel, otherwise, returns $\perp$. π_P^i may interact with a user oracle internally but such interactions are not revealed to the adversary due to our assumption on human communications.

This query allows the adversary to send any message to a specified non-user oracle and get its response in the binding or authenticated channel phase. It also allows the adversary to reboot a specified authenticator.

• $\mathsf{Authenticate}(\pi_C^i, M)$, for $C \in \mathcal{C}, i \in [N], M \in \mathcal{M}$.

This query returns $\perp$ if π_C^i has not set its binding state. It asks π_C^i to output the authenticated message $\widehat{M}$ (for the given input M) that it would send to an authenticator in the authenticated channel phase, then returns $\widehat{M}$.

This query allows the adversary to specify any message and get the corresponding authenticated message output by the specified client oracle in the authenticated channel phase. Note that we do not have a "verify" query to model an authenticator verifying the generated authenticated message because it actually models sending the generated message to the authenticator in the authenticated channel phase and getting the response, which is captured in our Send query.

• $\mathsf{Compromise}(\pi_C^i)$, for $C \in \mathcal{C}, i \in [N]$.

This query returns π_C^i's binding state. After this query, we say π_C^i was *compromised*.

This query allows the adversary to learn the binding state of a specified client oracle. Note that our model does not allow compromising an authenticator oracle because we assume authenticators are *tamper-proof*.

• $\mathsf{Corrupt}(U)$, for $U \in \mathcal{U}$.

This query returns user U's PIN pin_U. After this query, we say U was *corrupted*.

This query allows the adversary to learn a specified user's long-term secret PIN.

Partners. We say an authenticator oracle π_T^i and a client oracle π_C^j are each other's *partner* if there was a successful $\mathsf{Execute}(\cdot, \pi_T^i, \pi_C^j)$ query. Otherwise, we say they are each other's partner if they observe the same *session identifier* `sid` (in the binding phase) defined according to the protocol specifications and security goals, which is usually defined as a *subset* of the whole communication transcript. We also say π_C^j is T's partner (and hence T may have more than one partners). Note that as mentioned before, if an authenticator is rebooted then its power-up state is refreshed and all of its binding states (if any) are erased, i.e., the authenticator loses its existing partners after each reboot.

Advantage Measures. An adversary $\mathcal{A}$ against a PASACE protocol Π outputs $(\widehat{M}, x)$ in the end and has the following advantage measures:

• *Unforgeability (UF).* We define $\mathbf{Adv}_{\Pi}^{\mathsf{uf}}(\mathcal{A})$ as the probability that there exists an authenticator oracle π_T^i that accepted $(\widehat{M}, x)$ in the authenticated channel phase such that the following holds:

(1) $\widehat{M}$ was *not* output by T's partners (if any[4]) via previous $\mathsf{Authenticate}$ queries;

(2) No partners of T were compromised before π_T^i accepted $(\widehat{M}, x)$;

(3) T was set up by U and U was not corrupted before π_T^i accepted $(\widehat{M}, x)$.

The above captures the attacks in which the adversary successfully makes an authenticator accept a forged authenticated message, without corrupting the user who set up the authenticator or compromising any of the authenticator's partners. A PASACE protocol satisfying the above security notion should prevent an adversary from sending valid authenticated messages to the authenticator even if it stole the authenticator, unless the adversary corrupts the user that set up the authenticator or compromises any of the authenticator's partners. Note that the authenticated channels considered in this notion have only "weak" security, i.e., compromising one channel implies compromising all (to the same authenticator),

Then, to capture UF with *trusted binding* (UF-t) where the binding phase is also assumed to be run in a trusted way, we define its advantage measure $\mathbf{Adv}_{\Pi}^{\mathsf{uf\text{-}t}}(\mathcal{A})$ the same as $\mathbf{Adv}_{\Pi}^{\mathsf{uf}}(\mathcal{A})$ except that the adversary is *not* allowed to make $\mathsf{Connect}$ queries. Note that

[4]Recall that T's partners got refreshed every time $\mathsf{Reboot}(T)$ is queried.

unlike a trusted setup, here in the binding phase the adversary is still allowed to interact with the authenticator or the client (via Send queries) but it cannot interfere with an honest binding execution. Apparently, $\mathbf{Adv}_{\Pi}^{\mathsf{uf\text{-}t}}(\mathcal{A}) \leq \mathbf{Adv}_{\Pi}^{\mathsf{uf}}(\mathcal{A})$ and hence UF implies UF-t.

• *Strong Unforgeability (SUF).* We define $\mathbf{Adv}_{\Pi}^{\mathsf{suf}}(\mathcal{A})$ as the probability that there exists an authenticator oracle π_T^i that accepted $(\widehat{M}, x)$ in the authenticated channel phase such that the following holds:

(1) $\widehat{M}$ was *not* output by π_T^i's partner π_C^j (if any) via previous Authenticate queries;

(2) π_C^j was not compromised before π_T^i accepted $(\widehat{M}, x)$;

(3) T was set up by U and U was not corrupted before π_T^i set its binding state;

The above captures similar attacks considered in UF but in the client level, where the adversary is further allowed to compromise some of the target authenticator's partners (except the partner in the target authenticated channel) and corrupt the user even before the forged message was accepted (but after the authenticator set its binding state). The latter relaxation guarantees *forward secrecy* for authentication, which is not as strong as forward secrecy for confidentiality because breaking forward secrecy for authentication does not affect already authenticated messages but only affects future messages sent through an already established authenticated channel. Nevertheless, forward secrecy is still preferable. Besides, unlike UF, the authenticated channels considered in this notion have "strong" security, i.e., compromising one channel does not affect the other channels. It is not hard to see that SUF implies UF, i.e., $\mathbf{Adv}_{\Pi}^{\mathsf{uf}}(\mathcal{A}) \leq \mathbf{Adv}_{\Pi}^{\mathsf{suf}}(\mathcal{A})$. A PASACE protocol that is SUF secure should *not* have the same binding state for all partner client oracles (of the same authenticator), otherwise compromising any of them implies compromising all and hence the adversary can trivially win. More precisely, security can break down if two partner client oracles share the same binding state.

Similarly, one can define the advantage measure for SUF with trusted binding (SUF-t) $\mathbf{Adv}_{\Pi}^{\mathsf{suf\text{-}t}}(\mathcal{A})$. Obviously $\mathbf{Adv}_{\Pi}^{\mathsf{suf\text{-}t}}(\mathcal{A}) \leq \mathbf{Adv}_{\Pi}^{\mathsf{suf}}(\mathcal{A})$ and hence SUF implies SUF-t.

Furthermore, it is not hard to see that SUF-t implies UF-t.

Relations between Security Notions. As discussed above, Figure 5.2 shows the implication relations among our four defined notions. We will discuss examples in Section 5.5 to confirm the separation between UF and SUF-t, i.e., they do not imply each other.

$$\begin{tikzcd}
\text{SUF} \arrow[r, Rightarrow] \arrow[d, Rightarrow] & \text{UF} \arrow[d, Rightarrow] \arrow[dl, Rightarrow, shift left, "\times" marking] \\
\text{SUF-t} \arrow[r, Rightarrow] \arrow[ur, Rightarrow, shift left, "\times" marking] & \text{UF-t}
\end{tikzcd}$$

Figure 5.2: Relations between PASACE security notions.

5.4 The Client to Authenticator Protocol v2.0 and its Security

In this section, we present the FIDO Alliance's Client to Authenticator Protocol v2.0 (CTAP2) [38] of FIDO2 following our PASACE protocol syntax and show its provable security results based on our security model.

5.4.1 Protocol Description

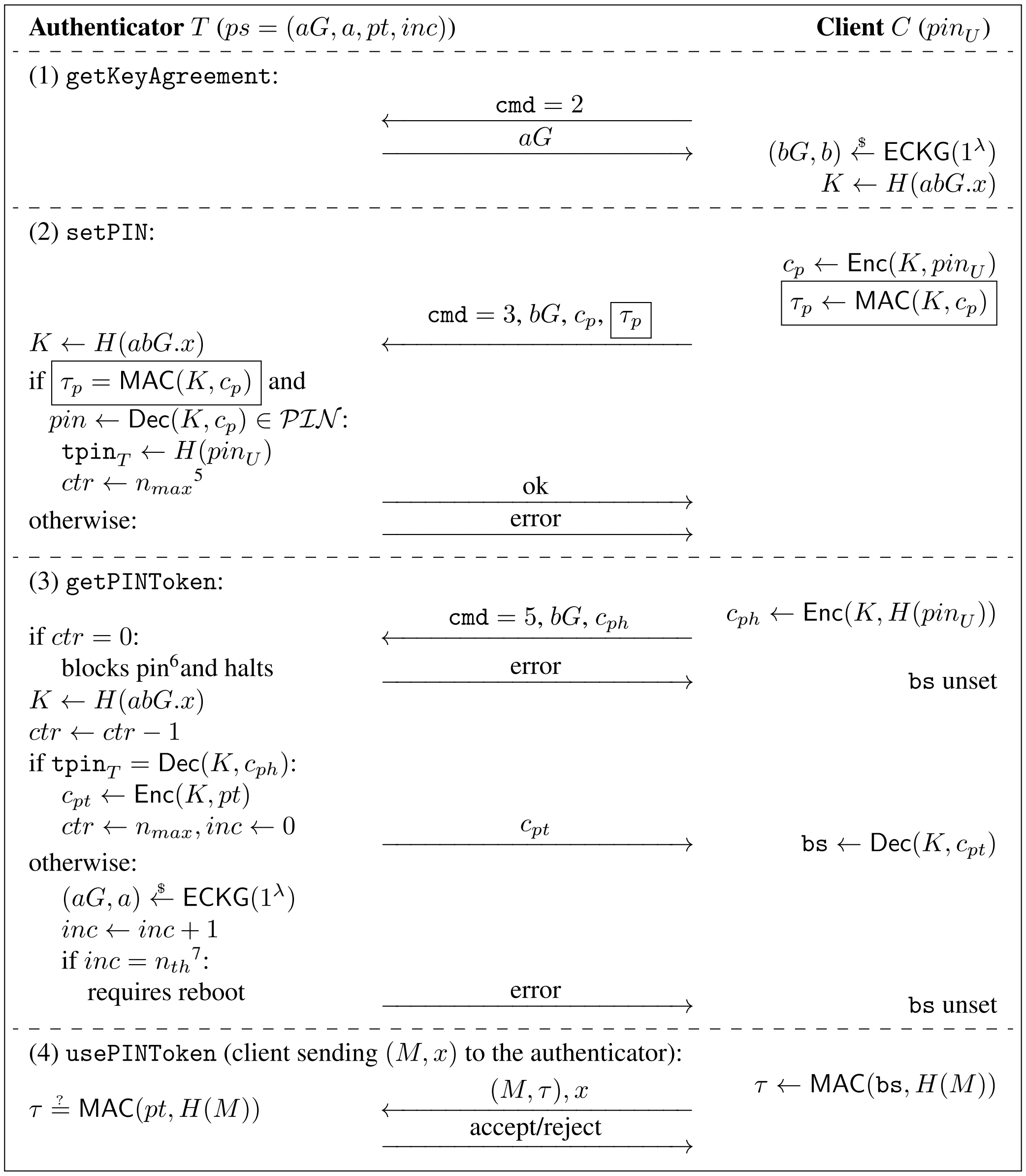
CTAP2 consists of several sub-protocols (requested with the corresponding sub-commands). For our analysis we focus on its core sub-protocols `getKeyAgreement`, `setPIN`, `getPINToken`, and `usePINToken` (ignoring `getRetries` and `changePIN`) (cf. Figure 1 in [38]). The PIN space $\mathcal{PIN}$ consists of 64-byte[8] strings (but a user PIN has low entropy, so $|\mathcal{PIN}|$ is small). The transformation function f is defined as the SHA-256 hash function (with output truncated to the first 128 bits) H and the corresponding transformed-PIN space $\mathcal{TPIN}$ is $\{H(pin)\}_{pin \in \mathcal{PIN}}$. Let ECKG denote the key generation function of the elliptic curve Diffie-Hellman (ECDH) scheme that takes as input 1^λ and outputs an elliptic curve public and secret key pair (aG, a), where G is an elliptic curve point

[5]By default $n_{max} = 8$.

[6]Once the authenticator blocks the pin, it needs to be reset to the factory default state (i.e., erasing all previous state) before further operations.

[7]By default $n_{th} = 3$.

[8]PINs memorized by users are of 4$\sim$63 bytes length in UTF-8 representation, padded with trailing 0 bytes.



Authenticator T ($ps = (aG, a, pt, inc)$)		**Client** C (pin_U)
(1) `getKeyAgreement`:		
	$\xleftarrow{\texttt{cmd} = 2}$	
	$\xrightarrow{aG}$	$(bG, b) \xleftarrow{\$} \mathsf{ECKG}(1^\lambda)$
		$K \leftarrow H(abG.x)$
(2) `setPIN`:		
		$c_p \leftarrow \mathsf{Enc}(K, pin_U)$
		$\boxed{\tau_p \leftarrow \mathsf{MAC}(K, c_p)}$
$K \leftarrow H(abG.x)$	$\xleftarrow{\texttt{cmd} = 3,\ bG,\ c_p,\ \boxed{\tau_p}}$	
if $\boxed{\tau_p = \mathsf{MAC}(K, c_p)}$ and		
$pin \leftarrow \mathsf{Dec}(K, c_p) \in \mathcal{PIN}$:		
$\texttt{tpin}_T \leftarrow H(pin_U)$		
$ctr \leftarrow n_{max}$[5]	$\xrightarrow{\text{ok}}$	
otherwise:	$\xrightarrow{\text{error}}$	
(3) `getPINToken`:		
	$\xleftarrow{\texttt{cmd} = 5,\ bG,\ c_{ph}}$	$c_{ph} \leftarrow \mathsf{Enc}(K, H(pin_U))$
if $ctr = 0$:		
blocks pin[6] and halts	$\xrightarrow{\text{error}}$	`bs` unset
$K \leftarrow H(abG.x)$		
$ctr \leftarrow ctr - 1$		
if $\texttt{tpin}_T = \mathsf{Dec}(K, c_{ph})$:		
$c_{pt} \leftarrow \mathsf{Enc}(K, pt)$		
$ctr \leftarrow n_{max}, inc \leftarrow 0$	$\xrightarrow{c_{pt}}$	$\texttt{bs} \leftarrow \mathsf{Dec}(K, c_{pt})$
otherwise:		
$(aG, a) \xleftarrow{\$} \mathsf{ECKG}(1^\lambda)$		
$inc \leftarrow inc + 1$		
if $inc = n_{th}$[7]:		
requires reboot	$\xrightarrow{\text{error}}$	`bs` unset
(4) `usePINToken` (client sending (M, x) to the authenticator):		
$\tau \stackrel{?}{=} \mathsf{MAC}(pt, H(M))$	$\xleftarrow{(M, \tau), x}$	$\tau \leftarrow \mathsf{MAC}(\texttt{bs}, H(M))$
	$\xrightarrow{\text{accept/reject}}$	

Figure 5.3: The CTAP2 protocol

generating a cyclic group $\mathbb{G}$ of prime order q and a is a random number in $\{1, \ldots, q-1\}$. (Note that CTAP2 uses the P-256 elliptic curve parameter set [112] for 128-bit security.) In CTAP2, the authenticator power-up state generation function ps_gen is defined as follows (where $\texttt{bl} = 128$ denotes the block size and pt is called the pinToken):

```
$\underline{\mathsf{ps\_gen}(1^\lambda)}$:
    $(aG, a) \xleftarrow{\$} \mathsf{ECKG}(1^\lambda), pt \xleftarrow{\$} \{0,1\}^{\texttt{bl}}, inc \leftarrow 0$
    return $(aG, a, pt, inc)$
```

Let $\mathsf{CBC}_0 = (\mathcal{K}, \mathsf{Enc}, \mathsf{Dec})$ denote the (deterministic) AES256-CBC encryption scheme with $\mathrm{IV} = 0$ and $\mathsf{MAC} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^{\mathtt{bl}}$ denote the HMAC-SHA256 message authentication code (with output truncated to the first 128 bits). The core sub-protocols of CTAP2 are presented in Figure 5.3, where in the beginning the client takes as input the user's PIN pin_U. It is not hard to see that CTAP2 is a PASACE protocol. More precisely, the authenticator setup phase corresponds to `getKeyAgreement`, `setPIN`, the binding phase corresponds to `getKeyAgreement`, `getPINToken`, and the authenticated channel phase corresponds to `usePINToken`.

5.4.2 Security Results

First, we notice that the MAC authentication (boxed in Figure 5.3) in `setPIN` is useless, i.e., it does not provide any authentication protection for a MITM attacker. A MITM attacker can pick its own ECDH key share to compute the shared key K that is used to generate valid ciphertexts and authentication tags. However, using the same key K for both encryption and authentication is considered bad practice and the resulting security guarantee is elusive for the CBC construction. Therefore, in our security analysis, we remove those redundant and problematic MAC operations and focus on the resulting simplified protocol (denoted by CTAP2*) which is more efficient and at least as secure as the original CTAP2.

It is not hard to see that CTAP2* is neither UF secure nor SUF-t secure (and hence SUF insecure). If the binding phase is not trusted, an attacker can impersonate the authenticator to get the PIN hash (i.e., transformed PIN) from the client which takes the user PIN as input. Then, the attacker is able to get pt from the authenticator. On the other hand, if any of the authenticator's partners is compromised, the attacker is able to get pt shared between them.

Now, we show that CTAP2* is UF-t secure. First, by modeling the AES-256 cipher as a PRF $E : \{0,1\}^{2\mathtt{bl}} \times \{0,1\}^{\mathtt{bl}} \to \{0,1\}^{\mathtt{bl}}$, we have the following theorem showing that CBC_0 is IND-1\$PA secure, with the proof in Appendix A.3.

Theorem 8. *For any PPT adversary $\mathcal{A}$, there exist a PPT adversary $\mathcal{B}$ such that:*

$$\mathbf{Adv}_{\mathsf{CBC}_0}^{\mathsf{ind\text{-}1\$pa}}(\mathcal{A}) \leq 2\mathbf{Adv}_{E}^{\mathsf{prf}}(\mathcal{B}) + \frac{\mu}{\mathtt{bl}}(\frac{\mu}{\mathtt{bl}} - 1)2^{-\mathtt{bl}},$$

where μ the total bit length of ciphertexts in response to all encryption oracle queries.

Then, by modeling the hash function H as a random oracle $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{\mathtt{bl}}$, we have the following theorem showing that CTAP2* is indeed UF-t secure, with the proof in Appendix A.4:

Theorem 9. *For any PPT adversary $\mathcal{A}$ making at most q_{E} $\mathsf{Execute}$ queries, q_{R} authenticator reboots, and $q_{\mathcal{H}}$ random oracle queries, there exist PPT adversaries $\mathcal{B}, \mathcal{C}, \mathcal{D}$ such that:*

$$\begin{aligned}\mathbf{Adv}_{\mathsf{CTAP2*}}^{\mathsf{uf\text{-}t}}(\mathcal{A}) \leq\ & q_{\mathsf{E}}\mathbf{Adv}_{\mathbb{G}}^{\mathsf{scdh}}(\mathcal{B}) + q_{\mathsf{E}}\mathbf{Adv}_{\mathsf{CBC}_0}^{\mathsf{ind\text{-}1\$pa}}(\mathcal{C}) + (|\mathcal{T}| + q_{\mathsf{R}})\mathbf{Adv}_{\mathsf{MAC}}^{\mathsf{euf\text{-}cma}}(\mathcal{D}) \\ & + \frac{nq_{\mathsf{E}}}{|\mathcal{PIN}|} + \frac{nq_{\mathsf{E}} + q_{\mathcal{H}}{}^2}{2^{\mathsf{bl}}},\end{aligned}$$

where $n = n_{th}$ if user undetectability is guaranteed or $n = n_{max}$ otherwise.

Note that in practical scenarios, if an authenticator is stolen by the attacker, n_{max} limits the maximum number of consecutive wrong PIN guesses before the authenticator blocks further interactions. On the other hand, if the target authenticator is not stolen (i.e., possessed by a user), then authenticator reboots imply user detectability. Therefore, n_{th} ($< n_{max}$) limits the maximum number of *undetectable* consecutive wrong PIN guesses after each honest binding execution.

Avoiding Authenticator Reboots Caused by Consecutive PIN Mismatches. As mentioned above, the n_{th} threshold is used for user detectability. To involve user interaction, CTAP2 requires the authenticator to reboot each time n_{th} consecutive PIN mismatches occur. Such reboots do not enhance security because the stored transformed PIN is not updated, but they could cause usability issue because each reboot invalidates all existing client-authenticator bindings. Therefore, we recommend it instead require a simple test of user presence (e.g., pressing a button) as well as resetting the inc counter to 0, when n_{th} consecutive PIN mismatches are detected.

Improving CTAP2*'s UF-t Security with Test of User Presence. Under the trusted binding assumption, we can make a simple modification to get rid of the only non-negligible $nq_{\mathsf{E}}/|\mathcal{PIN}|$ term (that accounts for online dictionary attacks) in the above security bound to improve the UF-t security of CTAP2* in the user-undetectable case. We only need to test the user presence (e.g., requiring the user to press a button on the authenticator) at the beginning of the binding phase. This helps prevent all malicious binding attempts because by assumption no honest bindings can be interrupted and now all other bindings become detectable. We argue that such test-of-user-presence overhead is quite small for CTAP2 because the user has to type his PIN into the client anyway. The security gain is considerable, because now *no* malicious binding attempts can happen and we get a neat negligible security bound.

5.5 The PIN-Based Authenticator Setup and Key Exchange Protocol and its Security

In this section, we propose a PASACE protocol called PIN-based Authenticator Setup and Key Exchange (PASKE) and prove its SUF security.

5.5.1 Protocol Description

PASKE consists of three sub-protocols (as shown in Figure 5.4) that respectively correspond to the three PASACE phases. It employs the same cryptographic primitives as CTAP2, as well as a PAKE protocol PAKE. Its authenticator power-up state generation function ps_gen is defined as follows (which does not generate the CTAP2 PINtoken pt):

```
$\underline{\mathsf{ps\_gen}(1^\lambda)}$:
    $(aG, a) \xleftarrow{\$} \mathsf{ECKG}(1^\lambda), inc \leftarrow 0$
    return $(aG, a, inc)$
```

[9]The user interaction could be pressing a button, which also needs to reset the inc counter to 0.

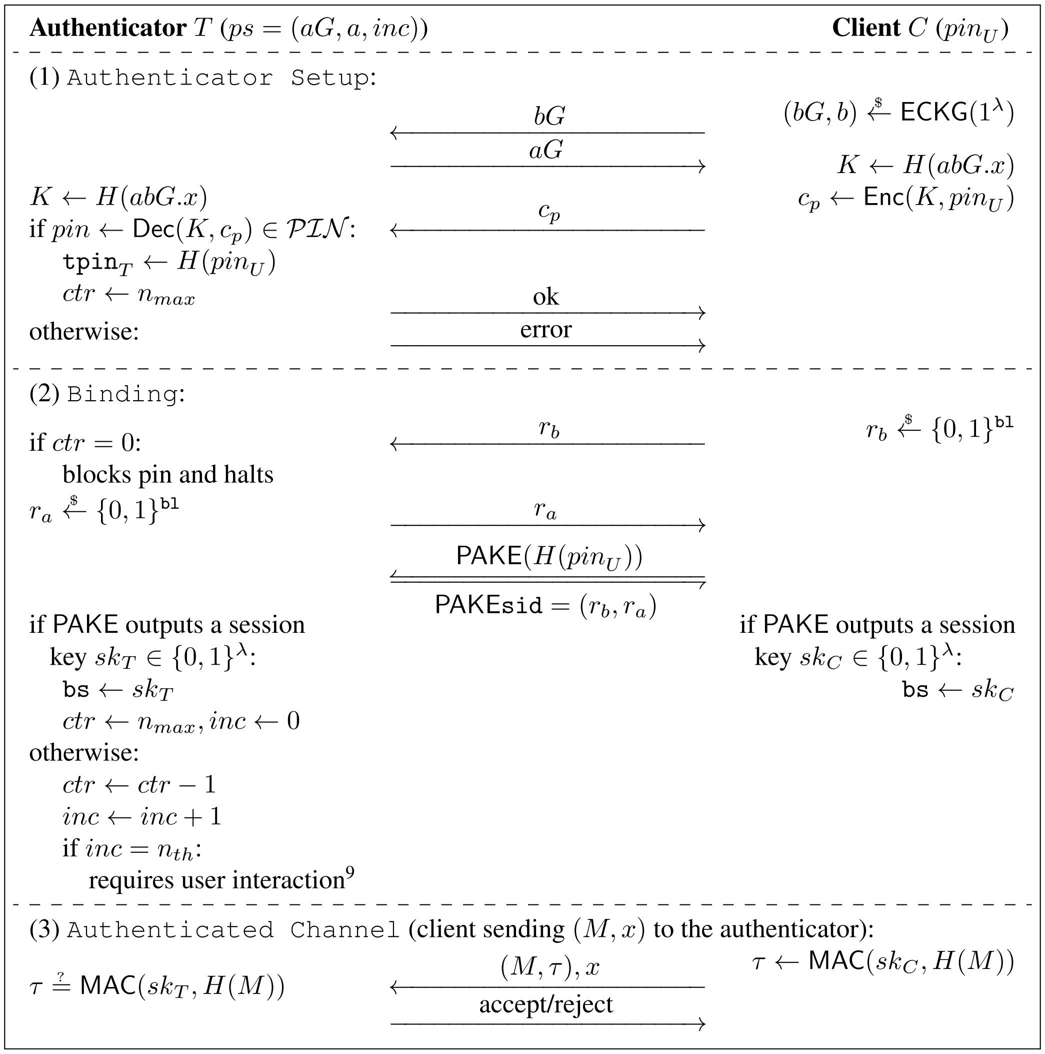

Authenticator T $(ps = (aG, a, inc))$
Client C (pin_U)
(1) Authenticator Setup:
$(bG, b) \xleftarrow{\$} \mathsf{ECKG}(1^\lambda)$
bG
aG
$K \leftarrow H(abG.x)$
$K \leftarrow H(abG.x)$
$c_p \leftarrow \mathsf{Enc}(K, pin_U)$
c_p
if $pin \leftarrow \mathsf{Dec}(K, c_p) \in \mathcal{PIN}$:
$\mathtt{tpin}_T \leftarrow H(pin_U)$
$ctr \leftarrow n_{max}$
ok
otherwise:
error
(2) Binding:
$r_b \xleftarrow{\$} \{0,1\}^{\mathtt{bl}}$
r_b
if $ctr = 0$:
blocks pin and halts
$r_a \xleftarrow{\$} \{0,1\}^{\mathtt{bl}}$
r_a
$\mathsf{PAKE}(H(pin_U))$
$\mathsf{PAKE}\mathtt{sid} = (r_b, r_a)$
if PAKE outputs a session key $sk_T \in \{0,1\}^\lambda$:
$\mathtt{bs} \leftarrow sk_T$
$ctr \leftarrow n_{max}, inc \leftarrow 0$
otherwise:
$ctr \leftarrow ctr - 1$
$inc \leftarrow inc + 1$
if $inc = n_{th}$:
requires user interaction[9]
if PAKE outputs a session key $sk_C \in \{0,1\}^\lambda$:
$\mathtt{bs} \leftarrow sk_C$
(3) Authenticated Channel (client sending (M, x) to the authenticator):
$\tau \leftarrow \mathsf{MAC}(sk_C, H(M))$
$(M, \tau), x$
$\tau \stackrel{?}{=} \mathsf{MAC}(sk_T, H(M))$
accept/reject

Figure 5.4: The PASKE protocol

5.5.2 Security Results

We show that PASKE is SUF secure. As discussed in Section 5.2, IND-1\$PA implies one-time IND-CPA, so according to Theorem 8 the one-time IND-CPA security of CBC_0 (of which the advantage is denoted by $\mathbf{Adv}^{\mathsf{ot\text{-}ind\text{-}cpa}}_{\mathsf{CBC}_0}(\mathcal{A})$) can also be reduced to (with negligible loss) the PRF security of E. The session identifier $\mathtt{sid}$ is defined as the concatenation of the client and authenticator's random nonces (r_b, r_a), which is also the session identifier of PAKE. We have the following theorem showing that PASKE achieves the strong SUF security in the random oracle and ideal cipher models, with the proof in Appendix A.5:

Theorem 10. *For any PPT adversary $\mathcal{A}$ making at most q_{E} $\mathsf{Execute}$ queries, q_{C} $\mathsf{Connect}$ queries, and $q_{\tilde{\mathsf{E}}}$ honest PAKE executions, there exist PPT adversaries $\mathcal{B}, \mathcal{C}, \mathcal{D}$ and environment $\mathcal{Z}$ such that:*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{suf}}_{\mathsf{PASKE}}(\mathcal{A}) \leq q_{\mathsf{E}}\mathbf{Adv}^{\mathsf{cdh}}_{\mathbb{G}}(\mathcal{B}) + q_{\mathsf{E}}\mathbf{Adv}^{\mathsf{ot\text{-}ind\text{-}cpa}}_{\mathsf{CBC}_0}(\mathcal{C}) + \mathbf{Adv}^{\mathsf{pake}}_{\mathsf{PAKE}}(\mathcal{S}_{\mathsf{PAKE}}, \mathcal{A}, \mathcal{Z}) \\ + |\mathcal{T}|N\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{MAC}}(\mathcal{D}) + \frac{nq_{\tilde{\mathsf{E}}} + q_{\mathsf{C}}}{|\mathcal{PIN}|} + \frac{(|\mathcal{U}|^2 + |\mathcal{T}|^2)N^2 + nq_{\tilde{\mathsf{E}}} + q_{\mathsf{C}} + {q_{\mathcal{H}}}^2}{2^{\mathsf{bl}}},\end{aligned}$$

where $n = n_{th}$ if user undetectability is guaranteed or $n = n_{max}$ otherwise.

Note that it is crucial for PAKE to guarantee *explicit* client authentication. Otherwise, the authenticator cannot detect wrong PIN guesses and decrease its ctr counter, which is used to prevent exhaustive PIN guesses.

Efficiency Comparison between PASKE and CTAP2. We purposely design our PASKE protocol following the original CTAP2 protocol such that the required modification is minimized. To achieve stronger security, PASKE introduces slightly more overhead. In particular, PASKE requires two and a half round trips (where PAKE takes one and a half) for its binding phase, while CTAP2 only runs two round trips. However, we argue that such overhead is very small especially when noticing that the client-authenticator communication is typically performed through USB or Bluetooth that do not involve significant network latency. Besides, we emphasize that the cryptographic primitives in PASKE could be instantiated with more efficient (but still secure) ones compared to those in CTAP2. For instance, one can use a simple one-time pad (with appropriate key expansion) instead of CBC_0 in the authenticator setup phase to achieve the same SUF security.

Notion Separation between UF and SUF-t. First, SUF-t does not imply UF. One can modify CTAP2* to generate an independent random pinToken for each partner client oracle. It is not hard to see that the resulting protocol is SUF-t secure but still not UF secure. On the other hand, UF does not imply SUF-t, either. One can modify PASKE to let the authenticator generate a global pinToken as with CTAP2* and send it (encrypted with the session key output by PAKE) to its partner in the end of the binding phase. The resulting protocol is clearly UF secure, but it is not SUF-t secure because of the same pinToken used by all partners.

5.6 Authenticator-Assisted Passwordless User Authentication

In this section, we define the syntax and security model for *Authenticator-assisted Passwordless User Authentication (APlUA)* protocols.

For simplicity, we do not consider certificates or certificate checks explicitly but assume any public verification key associated with an authenticator is supported by a *public key infrastructure (PKI)* and hence certified and bound to the authenticator's identity.

5.6.1 Protocol Syntax

An APlUA protocol is an interactive protocol among four parties: a user, an authenticator, a client, and a server. The user authenticates himself to the server via a client with the assistance of an authenticator (owned by the user and already registered to the server), instead of relying on a password.

$$\begin{tikzcd}
& \text{User} \arrow[dl, leftrightarrow, "\textcircled{1}"'] \arrow[dr, leftrightarrow, "\textcircled{2}"] & & \\
\text{Authenticator} \arrow[rr, leftrightarrow, "\textcircled{3}"] & & \text{Client} \arrow[r, leftrightarrow, "\textcircled{4}"] & \text{Server}
\end{tikzcd}$$

Figure 5.5: Communication Channels.

The possible communication channels are represented as double-headed arrows in Figure 5.5. Similar to PASACE protocols, we require messages sent to the user be *human-readable* and those sent by the user be *human-writable*.[10] We assume that the authenticator and client can distinguish messages received through a human communication channel ① ② from those received through other channels ③④.

[10]For APlUA protocols, we regard messages like authenticator displaying some user-defined messages (or simply blinking) as human-readable and pressing a button as human-writable.

The protocol is associated with the security parameter $\lambda \in \mathbb{N}_+$, a username space $\mathcal{UN} \subseteq \{0,1\}^*$, a server ID space $\mathcal{ID} \subseteq \{0,1\}^*$, a registration state space $\mathcal{RS} \subseteq \{0,1\}^*$, a message space $\mathcal{M} \subseteq \{0,1\}^* \setminus \{\varepsilon\}$, an auxiliary data space $\mathcal{X} \subseteq \{0,1\}^*$, and an efficient key generation algorithm Kg that takes as input 1^λ and outputs an attestation-verification key pair (ak, vk).

The protocol is also associated with a channel oracle $\mathcal{O}_{\mathsf{ch}}$ that establishes and uses *unidirectional* client-to-authenticator authenticated channels. It supports the following queries:

- $\mathcal{O}_{\mathsf{ch}}(\texttt{establish}, \mathit{chid} = (C, T))$ — this query is invoked by three parties, a user U, an authenticator T, and a client C: if T is (by default) not set up yet or T was set up by U, then records "T was set up by U" (if not recorded yet) and a C-to-T authenticated channel identified by a unique $\mathit{chid} = (C, T)$[11]; otherwise, returns $\perp$.

- $\mathcal{O}_{\mathsf{ch}}(\texttt{authenticate}, \mathit{chid} = (C, T), M)$, where $M \in \mathcal{M}$ — this query is invoked by the client C: records M as $\texttt{authenticated}$ with channel chid.

The protocol has two modes, *registration* and *authentication*. Its corresponding executions are referred to as the registration and authentication sessions. Each authentication session is associated with a *single* previous registration session which shares the same username[12].

In the beginning of a registration or authentication session, the user takes as input the intended server's ID $id \in \mathcal{ID}$ and a username $un \in \mathcal{UN}$, the authenticator takes as input its associated attestation key (generated by running $\mathsf{Kg}(1^\lambda)$), and the server takes as input its associated ID the authenticator's verification key. In an authentication session, the authenticator and server additionally takes as input its own registration state $\texttt{rs} \in \mathcal{RS}$ (set in the associated registration session). During the protocol's execution, each party (except the user) can keep states. In either mode, the user sends the first packet to start the session:

[11]We assume that there exists at most one channel between each client-authenticator pair.

[12]In practice, the registration and authentication sessions are associated with the same credential ID, since the same username may correspond to multiple credentials. Here to simplify our model we assume each username corresponds to one credential.

1) Registration. After receiving the first message, the client sets its target server ID $\mathtt{id} \in \mathcal{ID}$. In the end, the authenticator and server each sets its (perhaps different) registration state $\mathtt{rs} \in \mathcal{RS}$ on success or halts with $\mathtt{rs}$ unset on failure. If successful, we say the server *registered* the user's input username.

2) Authentication. After receiving the first message, the client sets its target server ID $\mathtt{id} \in \mathcal{ID}$ (which is initialized to the empty string ε at the beginning of each registration or authentication session). In the end, the server either *accepts* or *rejects*.

Correctness. Consider any user (with input un and id), authenticator, client, and server running an APlUA protocol in either mode. After receiving the first message, the client sets $\mathtt{id}$ equal to id. The authenticator only accepts messages through $\mathcal{O}_{\mathsf{ch}}(\mathtt{accept}, \cdot, \cdot, \cdot)$ queries. Assume an authenticated channel exists from the client to the authenticator. In a registration session, if un has not been registered to the server before, then in the end the registration succeeds, otherwise, the registration fails. In an authentication session, if the associated registration session succeeded and the same authenticator is used to run the authentication, then in the end the server accepts.

5.6.2 Security Model

We now formally define the security model for APlUA protocols.

Protocol Entities. The set of parties $\mathcal{P}$ consists of four disjoint type of parties: users $\mathcal{U}$, authenticators $\mathcal{T}$, clients $\mathcal{C}$, and servers $\mathcal{S}$, i.e., $|\mathcal{P}| = |\mathcal{U}| + |\mathcal{T}| + |\mathcal{C}| + |\mathcal{S}|$.

Session Oracles. Similar to PASACE protocols, each party $P \in \mathcal{P}$ is associated with a set of session oracles $\pi_P^1, \pi_P^2, \ldots$.

Communication Channel Security. For an APlUA execution, we assume the following security properties for the communication channels shown in Figure 5.5:

- Human communications (①②) are authenticated and private.

- Client-server communications (④) are not protected.

- Authenticator-client communications (③) are not protected, but as mentioned in the protocol syntax a client-to-authenticator authenticated channel can be established by querying the associated channel oracle $\mathcal{O}_{\mathsf{ch}}$.

Security Experiments. In the beginning of the experiments, run $\{(ak_T, vk_T) \overset{\$}{\leftarrow} \mathsf{Kg}(1^\lambda)\}_{T\in\mathcal{T}}$ to generate attestation-verification key pairs for all authenticators, run $\{id_S \in \mathcal{ID}\}_{S\in\mathcal{S}}$ to generate unique identities for all servers, and initialize global states of all parties and local states of all session oracles. We assume that the user sessions are *sequential*, i.e., a new user session oracle starts only if the user's previous session oracles have finished. Let $N \in \mathbb{N}_+$ denote the maximum number of APIUA protocol instances for each party. The channel oracle $\mathcal{O}_{\mathsf{ch}}$ is associated with a security level $level \in \{weak, strong\}$ that indicates the security level of the established channels. The adversary $\mathcal{A}$ is given all verification keys and server IDs, then interacts with session oracles via the following queries:

- $\mathsf{Channel}(U, T, C)$, for $U \in \mathcal{U}, T \in \mathcal{T}, C \in \mathcal{C}$.

This query asks U, T, C to invoke $\mathcal{O}_{\mathsf{ch}}(\mathtt{establish}, chid = (C, T))$ to establish a C-to-T authenticated channel (and set up T if necessary). (This query fails and returns $\perp$ if T was set up by a user other than U.) If U was corrupted (defined later), this query grants the adversary access to $\mathcal{O}_{\mathsf{ch}}(\mathtt{authenticate}, (C, T), \cdot)$. After this query, we say C is T's *channel partner*.

This query allows the adversary to establish an authenticated channel from a specified client oracle to a specified authenticator oracle with help of a specified user oracle, as well as set up the authenticator if necessary. If the user was corrupted, this query also grants the adversary access to the established channel.

- $\mathsf{Start}(\pi_U^l, \pi_T^i, \pi_C^j, S, un, \mathtt{mode})$, for $U \in \mathcal{U}, T \in \mathcal{T}, C \in \mathcal{C}, S \in \mathcal{S}, l, i, j \in [N], un \in \mathcal{UN}, \mathtt{mode} \in \{\mathtt{reg}, \mathtt{auth}\}$.

This query asks a user oracle π_U^l, which intends to register or authenticate to server S using username un via π_C^j with the assistance of π_T^i, to send the first message to π_C^j in a registration session if $\mathtt{mode} = \mathtt{reg}$ or in an authentication session if $\mathtt{mode} = \mathtt{auth}$, if T

was set up by U and there exists a C-to-T authenticated channel. The output of the client oracle π_C^j is returned to $\mathcal{A}$ instead of S.

This query allows the adversary to ask a specified user oracle to start a registration or authentication session towards a specified server using a specified username via a specified client oracle with the assistance of a specified authenticator oracle, then get the message output by the client oracle. This query also specifies an existing authenticated channel for later use, which may be used in multiple registration and authentication sessions.

• $\mathsf{Send}(\pi_P^i, m, n)$, for $P \in \mathcal{P} \setminus \mathcal{U}, i \in [N], m \in \{0,1\}^*, n \in \{1,3,4\}$.

This query sends m to a non-user oracle π_P^i through communication channel $\textcircled{n}$ as shown in Figure 5.5 and returns its response (if any). π_P^i may interact with a user oracle internally but such interactions are not revealed to the adversary due to our assumption on human communications. If $P \in \mathcal{C}$ and the response M is output to the authenticated channel $chid$, then also asks P to invoke $\mathcal{O}_{\mathsf{ch}}(\texttt{authenticate}, chid, M)$. If $P \in \mathcal{T}$ and $n = 3$, then m must be of the form $((C, P), M, x)$ for $C \in \mathcal{C}, M \in \mathcal{M}, x \in \mathcal{X}$, and π_P^i accepts the received message (M, x) if and only if: M was recorded as $\texttt{authenticated}$ with the specified channel (C, P) if $level = strong$ or with any channel $(\cdot, P)$ towards authenticator P if $level = weak$.

This query allows the adversary to send any message to a specified non-user oracle through a specified communication channel and get its response. This query also authenticates the output message sent through an existing authenticated channel. An authenticator oracle accepts only messages authenticated with the specified authenticated channel. Note that the adversary is not allowed to send messages to a client oracle via the human communication channel, in which case we consider the client oracle and the authenticated channel used by it are compromised as captured by the following query.

• $\mathsf{Compromise}(\pi_P^i$ or $chid)$, for $P \in \mathcal{C} \cup \mathcal{S}, i \in [N], chid \in \mathcal{C} \times \mathcal{T}$.

If the input is π_P^i, this query returns π_P^i's internal states. If the input is $chid = (C, T)$ that refers to an existing authenticated channel, this query grants the adversary access to

$\mathcal{O}_{\mathsf{ch}}(\texttt{authenticate}, chid, \cdot)$ for channel $chid$ if $level = strong$ or for all channels towards the same authenticator T if $level = weak$.

This query allows the adversary to learn all the internal states of a specified client or server oracle. It also allows the adversary to compromise an existing authenticated channel, this query grants the adversary access to the compromised channel for strong channels or to all authenticated channels towards the same authenticator for weak channels.

• $\mathsf{Corrupt}(U)$, for $U \in \mathcal{U}$.
If $level = weak$, This query grants the adversary access to $\mathcal{O}_{\mathsf{ch}}(\texttt{authenticate}, chid, \cdot)$ for all channels established by $\mathsf{Channel}(U, \cdot, \cdot)$. After this query, we say U was *corrupted*.

This query grants the adversary access to all authenticated channels related to a specified user for weak channels.

Partners. We say a server oracle and an authenticator oracle are each other's *partner* in a registration session if they observe the same *session identifier* $\texttt{sid}$, which is a subset of the communication transcript defined according to the protocol specifications and security goals. In an authentication session, we additionally require that the partnered session oracles are associated with the *same* registration session, i.e., their input registration states were set by partnered session oracles. We say an authenticator oracle and a client oracle are each other's *partner* if they observe the same (non-empty) messages transmitted through the authenticated channel.

Advantage Measures. An adversary $\mathcal{A}$ against an APlUA protocol Π has the following advantage measure:

• *User Authentication (UA).* We define $\mathbf{Adv}_{\Pi}^{\mathsf{ua}\text{-}level}(\mathcal{A})$ (recall that $level$ denotes the security level of the authenticated channels established by $\mathcal{O}_{\mathsf{ch}}$) as the probability that: there exists a server oracle π_S^k that accepts in an authentication session and its associated previous registration session oracle $\pi_S^{k'}$ $(k' < k)$ has a partner authenticator oracle $\pi_T^{i'}$ where T was set up by a user U, but the following does not hold:

(1) π_S^k has a unique authenticator partner π_T^i;

(2) π_T^i accepted a message through the human communication channel ①;

(3) π_T^i has a unique partner π_C^j and π_C^j has set `id` equal to id_S, if the following holds:

- U was not corrupted before π_S^k accepts, and
- no channels towards T were compromised before π_S^k accepts.

The above captures the attacks in which the adversary successfully impersonates a user to authenticate to a server that already registered the user's authenticator. For the above conditions, (1) implies that the registered authenticator must be used to authenticate to the server; (2) implies the authentication must be approved on the registered authenticator via the human communication channel; and (3) implies the authentication must go through a trusted client (i.e., a client that has an authenticated channel towards the authenticator) and the server that accepts must be the user's intended server if the adversary did not corrupt the user or compromise any of the registered authenticator's authenticated channels.

Note that by definition $\mathbf{Adv}_{\Pi}^{\mathsf{ua}\text{-}strong}(\mathcal{A}) \leq \mathbf{Adv}_{\Pi}^{\mathsf{ua}\text{-}weak}(\mathcal{A})$. That is, if an APlUA protocol is UA secure for weak channels, then it is also UA secure for strong channels.

• *Strong User Authentication (SUA).* We define $\mathbf{Adv}_{\Pi}^{\mathsf{sua}}(\mathcal{A})$ as the probability that $\mathcal{A}$ breaks UA (for either of the (1)$\sim$(3) conditions) or breaks the following:

(4) π_C^j is π_T^i's unique partner and π_C^j has set `id` equal to id_S, if the following holds:

- a $\mathsf{Start}(\pi_U^l, \pi_T^i, \pi_C^j, S, \cdot, \mathtt{auth})$ query was made,
- U was not corrupted before the (C, T) channel was established,
- the (C, T) channel was not compromised before π_S^k accepts, and
- during the authentication session of π_S^k, no Send queries were made to T through the human communication channel ①.

Unlike UA, the above is defined with respect to strong channels only, because it cannot be achieved with weak channels. The additional attacks captured by SUA are similar to those breaking condition (3) in UA, except that the adversary is further allowed to compromise some of the authenticated channels (and hence SUA cannot be achieved with weak channels) and corrupt the user even before the server accepts (but after the authenticated channel was established). However, the adversary is not allowed to send any message to the registered authenticator via the human communication channel, which in practice means the adversary does not *possess* the authenticator. This restriction is due to the following observation: it is impossible to prevent an adversary from authenticating to a server if it possesses the user's authenticator and meanwhile compromises any of the authenticator's authenticated channels. An APlUA protocol satisfying SUA additionally guarantees (compared to UA) that authenticating to a server through an uncompromised client using a user-possessed authenticator remains secure even if the adversary compromised other authenticated channels towards the registered authenticator.

Since the adversary can interact with the registered authenticator through a compromised authenticated channel, the authenticator has to send some messages to the user to help him detect malicious messages sent from unexpected (compromised) channels. This implies that the authenticator had better have a display in practice.

By definition, $\mathbf{Adv}_{\Pi}^{\mathsf{ua}\text{-}strong}(\mathcal{A}) \leq \mathbf{Adv}_{\Pi}^{\mathsf{sua}}(\mathcal{A})$ and hence SUA implies UA.

REMARK. Note that in our model only the authenticator has a secret, i.e., the attestation key, and the other parties essentially do not hold any secret (except a client may have private access to some authenticated channels). As a result, an APlUA protocol that is UA or SUA secure prevents many real-world attacks that compromise the parties (but not the authenticator which we assume is tamper-proof). For instance, it is immune to phishing attacks because the user does not have any private secret or information that can be revealed on the client. Besides, the server-in-the-middle attack presented in [108] is also ruled out because the adversary is already granted the ability to easily impersonate any server.

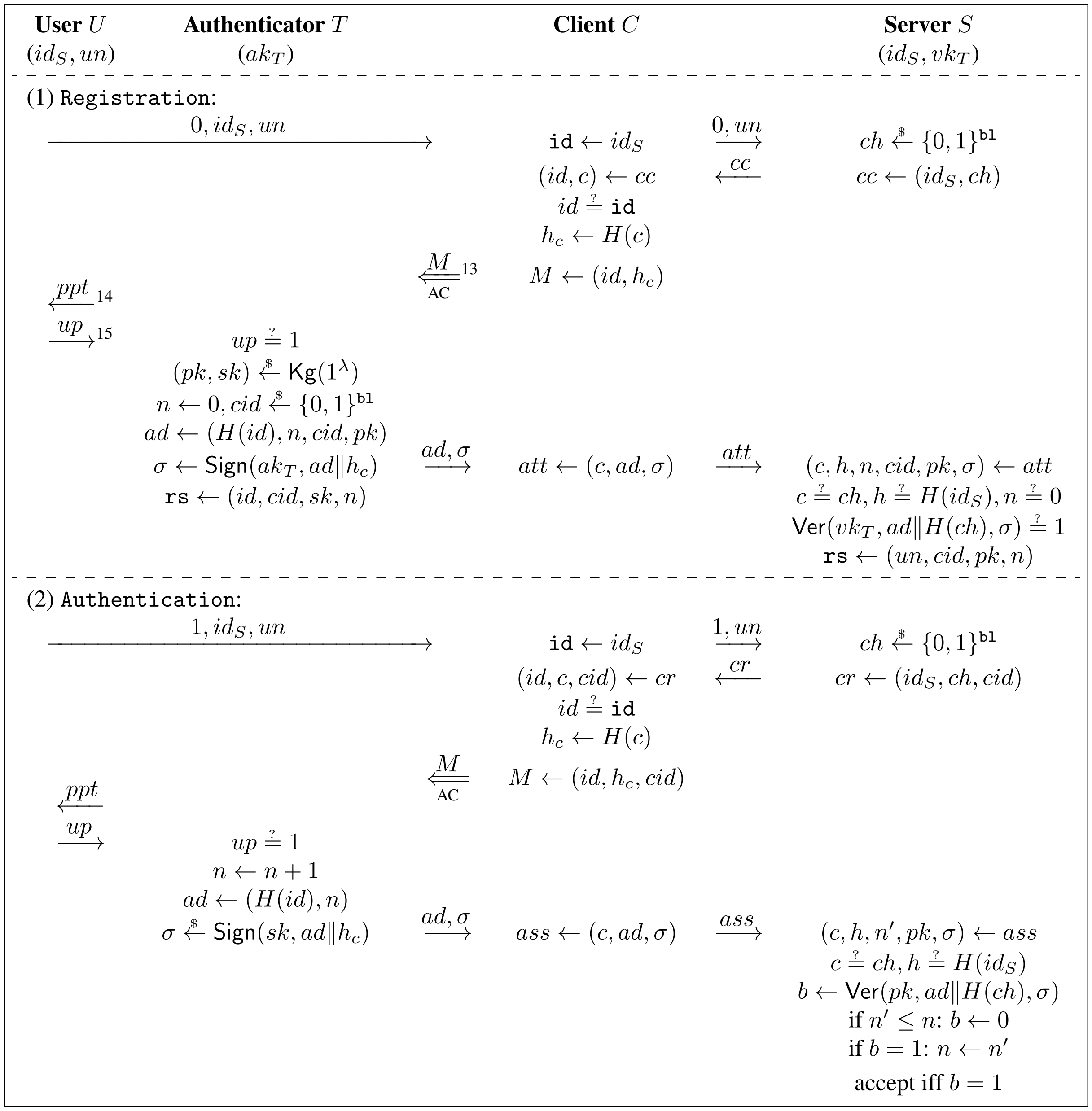

User U
(id_S, un)
Authenticator T
(ak_T)
Client C
Server S
(id_S, vk_T)
(1) Registration:
$0, id_S, un$
$\mathtt{id} \leftarrow id_S$
$0, un$
$ch \xleftarrow{\$} \{0,1\}^{\mathtt{bl}}$
$(id, c) \leftarrow cc$
cc
$cc \leftarrow (id_S, ch)$
$id \overset{?}{=} \mathtt{id}$
$h_c \leftarrow H(c)$
M
AC
13
$M \leftarrow (id, h_c)$
ppt 14
up 15
$up \overset{?}{=} 1$
$(pk, sk) \xleftarrow{\$} \mathsf{Kg}(1^\lambda)$
$n \leftarrow 0, cid \xleftarrow{\$} \{0,1\}^{\mathtt{bl}}$
$ad \leftarrow (H(id), n, cid, pk)$
$\sigma \leftarrow \mathsf{Sign}(ak_T, ad \| h_c)$
ad, σ
$att \leftarrow (c, ad, \sigma)$
att
$(c, h, n, cid, pk, \sigma) \leftarrow att$
$\mathtt{rs} \leftarrow (id, cid, sk, n)$
$c \overset{?}{=} ch, h \overset{?}{=} H(id_S), n \overset{?}{=} 0$
$\mathsf{Ver}(vk_T, ad \| H(ch), \sigma) \overset{?}{=} 1$
$\mathtt{rs} \leftarrow (un, cid, pk, n)$
(2) Authentication:
$1, id_S, un$
$\mathtt{id} \leftarrow id_S$
$1, un$
$ch \xleftarrow{\$} \{0,1\}^{\mathtt{bl}}$
$(id, c, cid) \leftarrow cr$
cr
$cr \leftarrow (id_S, ch, cid)$
$id \overset{?}{=} \mathtt{id}$
$h_c \leftarrow H(c)$
M
AC
$M \leftarrow (id, h_c, cid)$
ppt
up
$up \overset{?}{=} 1$
$n \leftarrow n + 1$
$ad \leftarrow (H(id), n)$
$\sigma \xleftarrow{\$} \mathsf{Sign}(sk, ad \| h_c)$
ad, σ
$ass \leftarrow (c, ad, \sigma)$
ass
$(c, h, n', pk, \sigma) \leftarrow ass$
$c \overset{?}{=} ch, h \overset{?}{=} H(id_S)$
$b \leftarrow \mathsf{Ver}(pk, ad \| H(ch), \sigma)$
if $n' \leq n$: $b \leftarrow 0$
if $b = 1$: $n \leftarrow n'$
accept iff $b = 1$

Figure 5.6: The WebAuthn protocol

5.7 The W3C Web Authentication Protocol and its Security

In this section, we present the W3C's Web Authentication (WebAuthn) protocol [37] of FIDO2 following our APlUA protocol syntax and show its provable security results based on our security model.

5.7.1 Protocol Description

WebAuthn supports two types of operations: `Registration` and `Authentication` (cf. Figure 1 and Figure 2 in [37]), respectively corresponding to the registration and authentication modes of an APlUA protocol. The username space $\mathcal{UN}$ consists of human-palatable strings for user accounts. The server ID space $\mathcal{ID}$ consists of effective domains (e.g., hostnames) of server URLs. The key generation algorithm Kg is defined as part of a signature scheme $\mathsf{Sig} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Ver})$. (Note that WebAuthn supports the RSASSA-PKCS1-v1_5 and RSASSA-PSS signature schemes [113].) Let H denote the SHA-256 hash function. The core cryptographic operations are presented in Figure 5.6. Obviously, WebAuthn is an APlUA protocol.

5.7.2 Security Results

We show that WebAuthn is UA secure with respect to weak channels. The session identifier `sid` is defined as the concatenation of the authenticator data and the signature (ad, σ). We have the following theorem showing that WebAuthn achieves UA security for weak channels, with the proof in Appendix A.6:

Theorem 11. *For any PPT adversary $\mathcal{A}$ that makes the authenticator oracles generate at most n_c credentials (i.e., public-secret key pairs generated by Kg in registration sessions), there exist PPT adversaries $\mathcal{B}, \mathcal{C}$ such that:*

$$\mathbf{Adv}^{\mathsf{ua}\text{-}weak}_{\mathsf{WebAuthn}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{coll}}_{H}(\mathcal{B}) + n_c \mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{Sig}}(\mathcal{C}) + \frac{|\mathcal{S}|N^2}{2^{\mathtt{bl}}}.$$

However, somewhat unexpectedly, WebAuthn is not SUA secure even equipped with strong channels. The reason is that in WebAuthn a user cannot detect if his authenticator ac-

[13]AC denotes the client-to-authenticator authenticated channel.

[14]*ppt* denotes the prompt generated by the authenticator, e.g., simple blinking or display of some client and server information.

[15]up denotes the bit indicating whether the user pressing a button on the authenticator as a test of user presence, which could be replaced by user verification uv for more powerful authenticators.

cepts messages through an unexpected (and perhaps compromised) channel. For instance, an attacker can trick a user to approve a malicious authentication request to a different server (that registered the same authenticator as the intended server) through a compromised channel, while the user might think the authentication goes through the specified uncompromised channel.

In the following section, we propose an augmented WebAuthn protocol that we call WebAuthn+ and prove its SUA security.

5.8 The WebAuthn+ Protocol and its Security

WebAuthn+ augments WebAuthn with simple operations as follows. After each authenticated channel is established, the user creates or chooses a recognizable unique name for the channel and inputs it to the client, which then sends the name to the authenticator through the established channel. Later when an authenticator seeks the human user for authentication approval, it first shows him the user-recognizable channel name. In this way, the user is able to detect malicious authentications and reject them if his authenticator accepts messages from an *unexpected* channel.

We have the following theorem showing that WebAuthn+ achieves SUA security, with the proof in Appendix A.7.

Theorem 12. *For any PPT adversary $\mathcal{A}$ that makes the authenticator oracles generate at most n_c credentials, there exist PPT adversaries $\mathcal{B}, \mathcal{C}$ such that:*

$$\mathbf{Adv}^{\mathsf{sua}}_{\text{WebAuthn+}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{coll}}_{H}(\mathcal{B}) + n_c \mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{Sig}}(\mathcal{C}) + \frac{|\mathcal{S}|N^2}{2^{\mathtt{bl}}}.$$

5.9 Composition

In this section, we show our composition theorem that states that a PASACE protocol can be securely composed with an APlUA protocol.

The protocol that composes a PASACE protocol Σ with an APlUA protocol Π is a "composed" protocol that replaces all queries to the "ideal" channel protocol $\mathcal{O}_{\mathsf{ch}}$ of Π with the corresponding operations of Σ. In particular, a $\mathcal{O}_{\mathsf{ch}}(\texttt{establish}, \mathit{chid} = (C, T))$ query is replaced with the first two phases of a new Σ instance among the client C, authenticator T, and user U that set up T (or a new user if T was not set up). On the other hand, a $\mathcal{O}_{\mathsf{ch}}(\texttt{authenticate}, \mathit{chid} = (C, T), M)$ query is replaced with asking C to output the authenticated message of M in the authenticated channel phase of the associated Σ instance.

We say a UF or UF-t security game generates weak authenticated channels, while a SUF or SUF-t security game generates strong authenticated channels. Let G_Σ be a game (i.e., security experiment) modeling the security for Σ that generates chl-level authenticated channels and G_Π a game associated with chl-level channels for Π, then $G_{\Sigma;\Pi}$ denotes the composed security game of G_Σ and G_Π, which essentially relays $\mathcal{O}_{\mathsf{ch}}$-related queries to G_Σ queries. The adversary wins in the composed game $G_{\Sigma;\Pi}$ if and only if it breaks the security of Π given queries of both G_Σ and G_Π. Note that the adversary $\mathcal{A}$ in the composed game inherits the same restrictions from G_Σ. For instance, if G_Σ refers to the UF-t security, then $\mathcal{A}$ cannot make $\mathsf{Connect}$ queries, i.e., the same trusted binding assumption is applied.

Our composition theorem is shown as follows, with the proof in Appendix A.8.

Theorem 13. *Let Σ be a secure PASACE protocol that generates chl-level channels and let Π be a secure APlUA protocol. If Π is secure for chl-level channels, then the composition $\Sigma; \Pi$ is secure with respect to experiment $G_{\Sigma;\Pi}$, i.e., for any PPT adversary $\mathcal{A}$, there exist PPT adversaries $\mathcal{B}, \mathcal{C}$ such that:*

$$\mathbf{Adv}_{\Sigma;\Pi}^{G_{\Sigma;\Pi}}(\mathcal{A}) \leq \mathbf{Adv}_{\Sigma}^{G_\Sigma}(\mathcal{B}) + \mathbf{Adv}_{\Pi}^{G_\Pi}(\mathcal{C}).$$

With our composition theorem, we derive that the overall FIDO2 achieves UF-t and UA security, while our new construction that integrates PASKE and WebAuthn+ achieves the strongest SUF and SUA security.

CHAPTER 6
CONCLUSION

In this thesis, we explore three interesting problems about secure communication and authentication and provide provable security analyses. We hope our results will help the practitioners understand the security guarantees and vulnerabilities of the important protocols and facilitate the design and deployment of more secure and efficient constructions.

Both our HAKE and FIDO2 analyses consider a human user as a separate party and capture human interactions with machines. We especially propose a concrete instantiation for the only-human HC function family based on previous work. We believe our work will promote further research and development in the area of human-oriented cryptography, which is promising to achieve stronger security by reducing trust on the client machines.

Our investigation of the most important low-latency secure channel establishment protocols (i.e., TLS 1.3 over TFO, QUIC over UDP, QUIC[TLS] over UDP) and passwordless authentication protocols (i.e., FIDO2) help clarify their precise security guarantees and limitations. Our models can be used as good references or guidelines for protocol designers and practitioners to better understand the important security aspects of novel secure communication and authentication protocols. Our proposed new constructions can serve as good candidates to achieve stronger security than existing protocols.

Appendices

APPENDIX A
PROOFS OF THEOREMS

A.1 Proof of Theorem 6

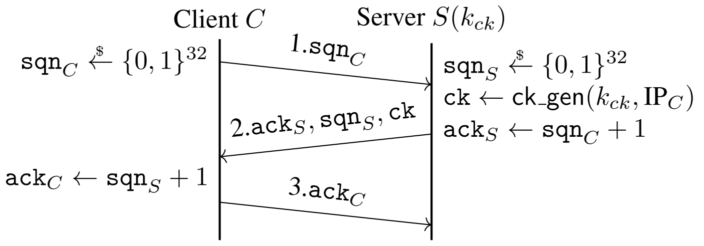

Client C
Server $S(k_{ck})$
$\mathtt{sqn}_C \xleftarrow{\$} \{0,1\}^{32}$
$1.\mathtt{sqn}_C$
$\mathtt{sqn}_S \xleftarrow{\$} \{0,1\}^{32}$
$\mathsf{ck} \leftarrow \mathsf{ck\_gen}(k_{ck}, \mathrm{IP}_C)$
$\mathtt{ack}_S \leftarrow \mathtt{sqn}_C + 1$
$2.\mathtt{ack}_S, \mathtt{sqn}_S, \mathsf{ck}$
$\mathtt{ack}_C \leftarrow \mathtt{sqn}_S + 1$
$3.\mathtt{ack}_C$

Figure A.1: TFO initial connection.

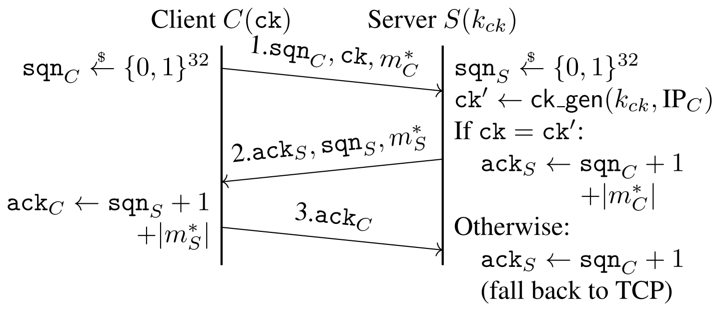

Client $C(\mathsf{ck})$
Server $S(k_{ck})$
$\mathtt{sqn}_C \xleftarrow{\$} \{0,1\}^{32}$
$1.\mathtt{sqn}_C, \mathsf{ck}, m_C^*$
$\mathtt{sqn}_S \xleftarrow{\$} \{0,1\}^{32}$
$\mathsf{ck}' \leftarrow \mathsf{ck\_gen}(k_{ck}, \mathrm{IP}_C)$
If $\mathsf{ck} = \mathsf{ck}'$:
$\mathtt{ack}_S \leftarrow \mathtt{sqn}_C + 1 + |m_C^*|$
$2.\mathtt{ack}_S, \mathtt{sqn}_S, m_S^*$
$\mathtt{ack}_C \leftarrow \mathtt{sqn}_S + 1 + |m_S^*|$
$3.\mathtt{ack}_C$
Otherwise:
$\mathtt{ack}_S \leftarrow \mathtt{sqn}_C + 1$
(fall back to TCP)

Figure A.2: TFO 0-RTT resumption connection. * indicates optional messages.

The TFO protocol specifications are shown in Fig. A.1 and Fig. A.2, where the server samples $k_{ck} \xleftarrow{\$} \{0,1\}^{\lambda}$ in the beginning and then generates cookies with $\mathsf{ck\_gen}$:

$\underline{\mathsf{ck\_gen}(k_{ck}, \mathrm{IP}_C)}$:

  return $F_{k_{ck}}(\mathrm{IP}_C \| \mathrm{IP}_S \| 0 \cdots 0)$

Proof. Consider a sequence of games $G_0, \ldots, G_{|\mathcal{S}|}$. G_0 is the real experiment for $\mathcal{A}$ and $G_{|\mathcal{S}|}$ uses random functions instead of the PRF F for all servers. The hybrid game G_i uses random functions for the first i servers and PRF for the last $|\mathcal{S}| - i$ servers. Denote $\Pr_i$ as the winning probability of $\mathcal{A}$ in G_i. By the PRF definition, for any $i \in [|\mathcal{S}|]$ there exists a PPT adversary $\mathcal{B}_i$ such that $|\Pr_{i-1} - \Pr_i| \leq \mathbf{Adv}_F^{\mathsf{prf}}(\mathcal{B}_i)$. Therefore, there exists a PPT adversary $\mathcal{B}$ such that $|\Pr_0 - \Pr_{|\mathcal{S}|}| \leq |\mathcal{S}|\mathbf{Adv}_F^{\mathsf{prf}}(\mathcal{B})$.

Now we only need to bound $\Pr_{|\mathcal{S}|}$ by considering two cases. 1) $\mathcal{A}$ wins by sending a valid ACK packet. In this case, $\mathcal{A}$ must have generated a valid $\mathtt{ack}_C$ by correctly guessing the target server's TCP sequence number $\mathtt{sqn}_S$. The winning probability of each guess is exactly $1/2^{|\mathsf{sqn}|}$. 2) $\mathcal{A}$ wins by sending a valid SYN packet in resumption sessions. In this case, $\mathcal{A}$ must have forged a valid TFO cookie ck. The winning probability of each forgery is exactly $1/2^n$ because the TFO cookie generation functions are independent and truly random. By applying a union bound on the q queries, we have $\Pr_{|\mathcal{S}|} \leq q/\min\{2^{|\mathsf{sqn}|}, 2^n\}$.

□

A.2 Proof of Theorem 7

Proof. Consider a sequence of games (i.e., experiments) and let $\Pr_i, i \geq 0$ denote the winning probability of $\mathcal{A}$ in **Game** i.

Game 0: This is the real experiment for $\mathcal{A}$, so $\Pr_0 = \mathbf{Adv}_{\text{UDP+QUIC[TLS]}}^{\mathsf{rst\text{-}auth}}(\mathcal{A})$.

Game 1: The challenger proceeds as before except it aborts if the connection IDs repeat for a party. Since the probability of $\mathtt{cid}$ collision for each party is at most $N^2/2^{|\mathtt{cid}|}$, by a union bound we have $|\Pr_0 - \Pr_1| \leq |\mathcal{P}|N^2/2^{|\mathtt{cid}|}$.

Game 2: The challenger proceeds as before except it replaces the PRFs F used for reset token generation with independent random functions f. By a hybrid argument similar to the proof of Theorem 6, there exists a PPT adversary $\mathcal{B}$ such that $|\Pr_1 - \Pr_2| \leq |\mathcal{P}|\mathbf{Adv}_F^{\mathsf{prf}}(\mathcal{B})$.

Game 3: The challenger proceeds as before except it replaces the encrypted pre-reset messages (via $\mathsf{Pack}(\cdot, \cdot, \mathtt{prst})$ queries) with encryption of independent random pre-reset messages. There exists a PPT adversary $\mathcal{C}$ against the Channel Security such that $|\Pr_2 - \Pr_3| \leq \mathbf{Adv}_{\text{UDP+QUIC[TLS]}}^{\mathsf{cs\text{-}1}}(\mathcal{C})$.

$\mathcal{C}$ can simulate Pack and $\mathsf{Deliver}$ queries with $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$ queries. For a $\mathsf{Pack}(\cdot, \cdot, \mathtt{prst})$ query, $\mathcal{C}$ generates two random reset tokens $x, y \xleftarrow{\$} \{0,1\}^n$ and uses them to construct two pre-reset messages. Then, $\mathcal{C}$ queries $\mathsf{Encrypt}$ with these pre-reset messages as challenge messages, uses the output ciphertext to form a pre-reset packet, and sends it to $\mathcal{A}$. For a $\mathsf{Pack}(\cdot, \cdot, \mathtt{rst})$ query, $\mathcal{C}$ returns the pre-reset message constructed from x. This perfectly simulates Game 2 in the left world or Game 3 in the right world. $\mathcal{C}$ outputs 1 if and only if $\mathcal{A}$ wins.

Now, in Game 3, the random pre-reset messages are independent from $\mathcal{A}$'s view and each guess is correct with probability $1/2^n$. By a union bound, we have $\Pr_3 \leq q_{\mathsf{rst}}/2^n$. □

A.3 Proof of Theorem 8

Proof. First, we replace the PRF E in both the left and right worlds with a purely random function $f : \{0,1\}^\lambda \to \{0,1\}^\lambda$. Note that it is straightforward to simulate either world of the IND-1\$PA experiment of CBC_0 with oracles E_k (for random k) or f. Therefore, there exists an efficient adversary $\mathcal{B}$ against the PRF security of E such that the game differences caused by replacing E with f are bounded by $2\mathbf{Adv}_E^{\mathsf{prf}}(\mathcal{B})$.

Let $\widetilde{\mathsf{CBC}}_0$ denote the CBC_0 encryption scheme of which the underlying PRF is replaced with a purely random function. We are left to show that the left and right worlds of the IND-1\$PA experiment of $\widetilde{\mathsf{CBC}}_0$ are computationally indistinguishable. It is sufficient to bound the probability of input collisions for f because the two worlds are perfectly indistinguishable if no collisions occur. Let p denote such collision probability. Since there are $\mu/\mathtt{bl}$ blocks in each world, by a union bound, we have $p \leq \mu/\mathtt{bl}(\mu/\mathtt{bl} - 1)2^{-\mathtt{bl}}$. We also refer to [114] (Lemma 18) for a more detailed security analysis of the randomized CBC scheme that also applies to our case. □

A.4 Proof of Theorem 9

Proof. Consider a sequence of games (i.e., experiments) and let $\Pr_i, i \geq 0$ denote the winning probability of $\mathcal{A}$ in **Game** i.

Game 0: This is the real experiment for $\mathcal{A}$, so $\Pr_0 = \mathbf{Adv}^{\mathsf{uf\text{-}t}}_{\mathsf{CTAP2*}}(\mathcal{A})$.

Game 1: The challenger proceeds as before except it replaces all shared keys $K = \mathcal{H}(abG.x)$ established in Execute queries with independent random values $\widetilde{K} \xleftarrow{\$} \{0,1\}^{\lambda}$. By a hybrid argument, there exists an efficient adversary $\mathcal{B}$ against the SCDH security of $\mathbb{G}$ such that $|\Pr_0 - \Pr_1| \leq q_{\mathsf{E}}\mathbf{Adv}^{\mathsf{scdh}}_{\mathbb{G}}(\mathcal{B})$.

To simulate a hybrid game, we embed the challenge group elements $\tilde{a}G, \tilde{b}G$ into the corresponding authenticator and client oracles and answer random oracle $\mathcal{H}$ queries via lazy sampling. Note that when receiving an arbitrary bG from $\mathcal{A}$ to the authenticator oracle via a Send query, we check if $\tilde{a}bG.x$ has already been queried to the random oracle $\mathcal{H}$ using the verification oracle $\mathcal{O}_{\tilde{a}}(\cdot,\cdot)$. This is sufficient to answer $\mathcal{H}$ queries consistently and set the resulting shared keys accordingly. Before setting $K = \mathcal{H}(\tilde{a}\tilde{b}G.x)$ to $\widetilde{K}$, we also need to check if $\tilde{a}\tilde{b}G$ has already been queried: if so we can win the SCDH game, otherwise we simulate $\mathcal{A}$'s view perfectly.

Game 2: The challenger proceeds as before except it replaces all transmitted ciphertexts $c_p = \mathsf{Enc}(\widetilde{K}, 0, pin)$, $c_{ph} = \mathsf{Enc}(\widetilde{K}, 0, \mathcal{H}(pin))$, $c_{pt} = \mathsf{Enc}(\widetilde{K}, 0, pt)$ in Execute queries with $\tilde{c}_p \leftarrow \mathsf{Enc}(\widetilde{K}, 0, \widetilde{pin})$, $\tilde{c}_{ph} \leftarrow \mathsf{Enc}(\widetilde{K}, 0, \mathcal{H}(\widetilde{pin}))$, $\tilde{c}_{pt} \leftarrow \mathsf{Enc}(\widetilde{K}, 0, \widetilde{pt})$, where $\widetilde{pin} \xleftarrow{\$} \mathcal{PIN}, \widetilde{pt} \xleftarrow{\$} \{0,1\}^{\lambda}$ are independent random values. By a hybrid argument, there exists an efficient adversary $\mathcal{C}$ against the IND-1\$PA security of CBC_0 such that $|\Pr_1 - \Pr_2| \leq q_{\mathsf{E}}\mathbf{Adv}^{\mathsf{ind\text{-}1\$pa}}_{\mathsf{CBC}_0}(\mathcal{C})$.

To simulate a pair of consecutive hybrid games G_k, G_{k+1} (e.g., replacing the transmitted ciphertexts in $\mathsf{Execute}(\pi^l_U, \pi^i_T, \pi^j_C)$), we sample an independent random PIN $\widetilde{pin} \xleftarrow{\$} \mathcal{PIN}$ and query the IND-1\$PA encryption oracles of CBC_0 as follows. First, query $c^* \leftarrow \mathcal{O}_{\mathsf{LR}}(pin_U, \widetilde{pin})$ if the considered Execute query involves the authenticator setup phase.

Then, if $pin_U \neq \widetilde{pin}$, query $(m_0, m_1, c_b) \xleftarrow{\$} \mathcal{O}_{\$\mathsf{LR}}(1, \mathtt{rand})$; otherwise (if $pin_U = \widetilde{pin}$), query $(m, c) \xleftarrow{\$} \mathcal{O}_{\$\mathsf{LR}}(1, \mathtt{same})$ and let $m_0 = m_1 = m$ and $c_b = c$. Finally, query $(m'_0, m'_1, c'_b) \xleftarrow{\$} \mathcal{O}_{\$\mathsf{LR}}(1, \mathtt{rand})$. Now, we set $c_p \leftarrow c^*$, $c_{ph} \leftarrow c_b$, $c_{pt} \leftarrow c'_b$, $pt \leftarrow m'_0$. It is not hard to see that, in the left world, the above ciphertexts and pt are equal to the original ones in game G_k while, in the right world, they are equal to the replaced ciphertexts that encrypt random values in game G_{k+1}. Note that we simulate the random oracle $\mathcal{H}$ via lazy sampling as before except that we fix $\mathcal{H}(pin_U) \leftarrow m_0$ and $\mathcal{H}(\widetilde{pin}) \leftarrow m_1$ (or $\mathcal{H}(pin) \leftarrow m$ if $pin_U = \widetilde{pin}$) in the beginning.

Game 3: The challenger proceeds as before except it aborts if some authenticator oracle received a malicious c_{ph} from $\mathcal{A}$ (via a Send query) and the decrypted message matches its transformed PIN `tpin`. Note that after Game 2, the authenticators' transformed PINs and the clients' input PINs are independent from $\mathcal{A}$'s view. Since each $\mathsf{Execute}$ query allows $\mathcal{A}$ to make at most n_{max} guesses (or n_{th} guesses without requiring authenticator reboots) about the corresponding authenticator's `tpin` and each guess succeeds with probability at most $1/|\mathcal{PIN}| + 1/2^{\mathsf{bl}}$ (where $1/2^{\mathsf{bl}}$ accounts for the chance that two different PINs result in the same $\mathcal{H}$ output), by a union bound we have $|\Pr_2 - \Pr_3| \leq nq_{\mathsf{E}}(1/|\mathcal{PIN}| + 1/2^{\mathsf{bl}}) = nq_{\mathsf{E}}/|\mathcal{PIN}| + nq_{\mathsf{E}}/2^{\mathsf{bl}}$.

Now, Game 3 can be simulated by an efficient adversary $\mathcal{D}$ against the EUF-CMA security of the MAC scheme MAC. $\mathcal{D}$ first guesses the accepting authenticator T and its reboot cycle with probability at least $1/(|\mathcal{T}| + q_{\mathsf{R}})$ (because each authenticator reboot generates an independent random pt), then simulates the game with the MAC oracle of MAC. This simulation is perfect except when collisions occur in the random oracle queries (used to get the message digest before signing), which happens with probability at most $q_{\mathcal{H}}{}^2/2^{\mathsf{bl}}$. Therefore, $\Pr_3 \leq (|\mathcal{T}| + q_{\mathsf{R}})\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{MAC}}(\mathcal{D}) + q_{\mathcal{H}}{}^2/2^{\mathsf{bl}}$. $\square$

A.5 Proof of Theorem 10

Proof. Consider a sequence of games (i.e., experiments) and let $\Pr_i, i \geq 0$ denote the winning probability of $\mathcal{A}$ in **Game** i.

Game 0: This is the real experiment for $\mathcal{A}$, so $\Pr_0 = \mathbf{Adv}^{\mathsf{suf}}_{\mathsf{PASKE}}(\mathcal{A})$.

Game 1: The challenger proceeds as before except it replaces all shared keys $K = \mathcal{H}(abG.x)$ established in Execute queries with independent random values $\widetilde{K} \xleftarrow{\$} \{0,1\}^\lambda$. Since ECDH is executed only in the trusted authenticator setup phase, our proof no longer requires the SCDH assumption. Similar to the corresponding proof in Appendix A.4, there exists an efficient adversary $\mathcal{B}$ against the CDH security of $\mathbb{G}$ such that $|\Pr_0 - \Pr_1| \leq q_{\mathsf{E}}\mathbf{Adv}^{\mathsf{cdh}}_{\mathbb{G}}(\mathcal{B})$.

Game 2: The challenger proceeds as before except it replaces all transmitted ciphertexts $c_p = \mathsf{Enc}(\widetilde{K}, 0, pin)$ in Execute queries with $\tilde{c}_p \leftarrow \mathsf{Enc}(\widetilde{K}, 0, \widetilde{pin})$ for an independent random $\widetilde{pin} \xleftarrow{\$} \mathcal{PIN}$. By a hybrid argument, there exists an efficient adversary $\mathcal{C}$ against the one-time IND-CPA security of CBC_0 such that $|\Pr_1 - \Pr_2| \leq q_{\mathsf{E}}\mathbf{Adv}^{\mathsf{ot\text{-}ind\text{-}cpa}}_{\mathsf{CBC}_0}(\mathcal{C})$.

To simulate a pair of consecutive hybrid games G_k, G_{k+1} (e.g., replacing c_p with $\tilde{c}_p$ in $\mathsf{Execute}(\pi^l_U, \pi^i_T, \pi^j_C)$), we sample an independent random PIN $\widetilde{pin} \xleftarrow{\$} \mathcal{PIN}$, query $c_b \leftarrow \mathcal{O}_{\mathsf{LR}}(pin_U, \widetilde{pin})$, and set $c_p \leftarrow c_b$, which simulates the games perfectly.

Game 3: The challenger proceeds as before except it aborts if there exist two sessions with the same session identifier. Recall that the nonces r_a, r_b are chosen independently at random by honest parties, so any two session identifiers collide with probability at most $1/2^{\mathsf{bl}}$ on each half. Since there are at most $|\mathcal{U}|N$ sessions that involves clients and at most $|\mathcal{T}|N$ sessions that involves authenticators, by a union bound we have $|\Pr_2 - \Pr_3| \leq (|\mathcal{U}|^2 + |\mathcal{T}|^2)N^2/2^{\mathsf{bl}}$.

Game 4: The challenger proceeds as before except it replaces PAKE with the ideal functionality $\hat{\mathcal{F}}^{\mathsf{CA}}_{\mathsf{PAKE}}$ and uses $\mathcal{S}_{\mathsf{PAKE}}$ to simulate interactions with PAKE. By definition, there exists an efficient environment $\mathcal{Z}$ against the UC security of PAKE with ideal functionality $\hat{\mathcal{F}}_{\mathsf{PAKE}}^{\mathsf{CA}}$ in the random oracle and ideal cipher models such that $|\Pr_3 - \Pr_4| \leq \mathbf{Adv}_{\mathsf{PAKE}}^{\mathsf{pake}}(\mathcal{S}_{\mathsf{PAKE}}, \mathcal{A}, \mathcal{Z})$.

Game 5: The challenger proceeds as before except it aborts if some session got compromised, i.e., $\mathcal{S}_{\mathsf{PAKE}}$ made a `TestPwd` query and guessed the password correctly. Note that after Game 4, the authenticators' transformed PINs and the clients' input PINs are independent from $\mathcal{A}$'s view. Furthermore, authenticators can detect all wrong password guesses because $\hat{\mathcal{F}}_{\mathsf{PAKE}}^{\mathsf{CA}}$ includes client authentication. Since each honest PAKE execution allows $\mathcal{A}$ to make at most n_{max} guesses (or n_{th} guesses without requiring authenticator reboots) about the password (i.e., $\mathcal{H}(pin)$) by interacting with an authenticator and $\mathcal{A}$ has to make a Connect query before guessing the password by interacting with a client, in total $\mathcal{A}$ can make at most $nq_{\tilde{\mathsf{E}}} + q_{\mathsf{C}}$ guesses. Then, because each guess succeeds with probability at most $1/|\mathcal{PIN}| + 1/2^{\mathsf{bl}}$, by a union bound we have $|\Pr_4 - \Pr_5| \leq (nq_{\tilde{\mathsf{E}}} + q_{\mathsf{C}})(1/|\mathcal{PIN}| + 1/2^{\mathsf{bl}}) = (nq_{\tilde{\mathsf{E}}} + q_{\mathsf{C}})/|\mathcal{PIN}| + (nq_{\tilde{\mathsf{E}}} + q_{\mathsf{C}})/2^{\mathsf{bl}}$.

Now, Game 5 can be simulated by an efficient adversary $\mathcal{D}$ against the EUF-CMA security of the MAC scheme MAC. $\mathcal{D}$ first guesses the accepting authenticator oracle π_T^i with probability at least $1/|\mathcal{T}|N$, then simulates the game with the MAC oracle of MAC. Recall that $\hat{\mathcal{F}}_{\mathsf{PAKE}}^{\mathsf{CA}}$ guarantees client authentication, i.e., π_T^i receives a random key (in an uncompromised session) only if there is a partner client oracle that received the same key. This ensures that $\mathcal{D}$ can simulate the MAC operations of π_T^i and its partner perfectly if no random oracle collisions occur. Therefore, we have $\Pr_5 \leq |\mathcal{T}|N\mathbf{Adv}_{\mathsf{MAC}}^{\mathsf{euf\text{-}cma}}(\mathcal{D}) + {q_{\mathcal{H}}}^2/2^{\mathsf{bl}}$. □

A.6 Proof of Theorem 11

Proof. Consider a sequence of games (i.e., experiments) and let $\Pr_i, i \geq 0$ denote the winning probability of $\mathcal{A}$ in **Game** i.

Game 0: This is the real experiment for $\mathcal{A}$, so $\Pr_0 = \mathbf{Adv}_{\text{WebAuthn}}^{\mathsf{ua}\text{-}weak}(\mathcal{A})$.

Game 1: The challenger proceeds as before except it aborts if a hash collision occurs. By definition, there exists a PPT adversary $\mathcal{B}$ against the collision-resistance security of H such that $|\Pr_0 - \Pr_1| \leq \mathbf{Adv}_H^{\mathsf{coll}}(\mathcal{B})$.

Game 2: The challenger proceeds as before except it aborts if there exists a server that generates two identical challenges in authentication sessions. By a union bound, we have $|\Pr_1 - \Pr_2| \leq |\mathcal{S}|N^2/2^{\mathsf{bl}}$.

Game 3: The challenger proceeds as before except it aborts if the first condition in UA does not hold, i.e., π_S^k does not have a unique partner π_T^i. This game can be simulated by a PPT adversary $\mathcal{C}$ against the EUF-CMA security of the signature scheme Sig as follows.

$\mathcal{C}$ first guesses π_S^k's credential with probability at least $1/n_c$, then simulates the game by answering all queries related to that credential with the signing oracle and public key of Sig. There are two cases, π_S^k has two partners or no partner. Because the signature counter n is incremented for every new session, π_S^k cannot have two partners using the same credential, except the same credential used by π_S^k was generated in another registration session. If this happens, $\mathcal{C}$ can trivially forge a signature to a new message as it knows the signing key. Otherwise, recall that partners in authentication sessions must refer to the same registration session and hence the same credential, so π_S^k cannot have two partners with different credentials. On the other hand, if π_S^k has no partner but still accepts in an authentication session, then $\mathcal{C}$ has forged a valid signature and wins the EUF-CMA game. Therefore, we have $|\Pr_2 - \Pr_3| \leq n_c \mathbf{Adv}_{\mathsf{Sig}}^{\mathsf{euf\text{-}cma}}(\mathcal{C})$.

Now, in Game 3 π_S^k has a unique partner π_T^i. Referring to Figure 5.6 for the authentication procedures of WebAuthn, it is obvious that the second condition in UA holds, i.e., π_T^i accepted a message through the human communication channel ①. Moreover, π_T^i must have accepted an authenticated message M through an authenticated channel. Since the adversary does not have access to any of the authenticated channels towards T, with weak channels M must be authenticated by a client oracle π_C^j. Note that M consists of the unique server ID and a challenge (that is also unique), so π_C^j is unique and the third condition in

UA also holds. Therefore, $\Pr_3 = 0$. $\square$

A.7 Proof of Theorem 12

Proof. The proof proceeds in the same way as the proof of Theorem 11 in Appendix A.6, except that it relies on strong channels and the augmented user detectability to show that the extra condition (4) in SUA also holds in the final game. More precisely, in the final game consider that π_S^k has a unique partner π_T^i and a $\mathsf{Start}(\pi_U^l, \pi_T^i, \pi_C^j, S, \cdot, \mathtt{auth})$ query was made. Since the adversary does not have access to the (C, T) channel or the human communication channel to π_T^i, for the authentication to succeed, π_U^l must approve it. Now with strong channels, due to the augmented user detectability, π_T^i must receive an authenticated message from the expected (C, T) channel authenticated by a session oracle of C. Recall that the user sessions are assumed to be sequential, π_C^j is the only active client oracle that can send an authenticated message to π_T^i through that channel, and hence π_C^j is π_T^i's unique client partner. Before sending the authenticated message, π_C^j must check that $\mathtt{id}$ equals id_S. $\square$

A.8 Proof of Theorem 13

Proof. Consider a sequence of games (i.e., experiments) and let $\Pr_i, i \geq 0$ denote the winning probability of $\mathcal{A}$ in **Game** i.

Game 0: This is the real experiment for $\mathcal{A}$, so $\Pr_0 = \mathbf{Adv}_{\Sigma;\Pi}^{G_{\Sigma;\Pi}}(\mathcal{A})$.

Game 1: The challenger proceeds as before except it aborts if the G_Σ security is broken, i.e., there exists an authenticator oracle that accepted a forged authenticated message. Note for any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ against the G_Σ security that can simulate all $G_{\Sigma;\Pi}$ queries perfectly. Therefore, $|\Pr_0 - \Pr_1| \leq \mathbf{Adv}_{\Sigma}^{G_\Sigma}(\mathcal{B})$.

Now Game 1 can be simulated by a PPT adversary $\mathcal{C}$ against the G_Π security. $\mathcal{C}$ samples all the secret PINs and authenticator power-up states and only uses authenticated channels established with its $\mathcal{O}_{\mathsf{ch}}$ queries for the G_Π subgame, i.e., $\mathcal{C}$ does not actually relay $\mathcal{O}_{\mathsf{ch}}$ queries to the simulated G_Σ queries in the composed game but only simulates such effect. By the design of the queries in our APlUA security experiments, such simulation is perfect given that the G_Σ security of Game 1 cannot be broken. Therefore, $\mathcal{A}$ wins implies $\mathcal{C}$ wins and we have $\Pr_1 \leq \mathbf{Adv}_{\Pi}^{G_\Pi}(\mathcal{C})$. $\square$

APPENDIX B
QUIC AND TLS 1.3'S STATEFUL AEAD SCHEMES AND THEIR SECURITY

In this chapter, we describe the stateful AEAD schemes used in QUIC and TLS and prove their security. We refer to [115] for the syntax and security definitions of a nonce-based AEAD scheme.

B.1 QUIC's Stateful AEAD Scheme and its Security

First, we show QUIC's stateful encryption scheme $\mathsf{sAEAD}_{\text{QUIC}}$ constructed from a nonce-based AEAD scheme $\mathsf{AEAD} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as follows. We also refer to [115] for the syntax and security definitions of a nonce-based AEAD scheme.

```
$\underline{\mathsf{sGen}()}$:
    $k_e \xleftarrow{\$} \mathsf{Gen}(), k_m \xleftarrow{\$} \{0,1\}^{32}$
    $(st_e, st_d) \leftarrow (\varnothing, \perp)$
    return $(k_e, k_m)$

$\underline{\mathsf{sDec}(k, ad, ct, st_d)}$:
    $(k_e, k_m) \leftarrow k$
    $(\mathtt{cid}, \mathtt{sqn}) \leftarrow ad$
    $m \leftarrow \mathsf{Dec}(k_e, k_m \| \mathtt{sqn}, ad, ct)$
    return $(m, \perp)$

$\underline{\mathsf{sEnc}(k, ad, m, st_e)}$:
    $(k_e, k_m) \leftarrow k$
    $(\mathtt{cid}, \mathtt{sqn}) \leftarrow ad$
    if $\mathtt{sqn} \in st_e$,
        return $(\perp, st_e)$
    $c \leftarrow \mathsf{Enc}(k_e, k_m \| \mathtt{sqn}, ad, m)$
    $st_e \leftarrow st_e \cup \{\mathtt{sqn}\}$
    return $(c, st_e)$
```

Note that $\mathsf{sAEAD}_{\text{QUIC}}$ uses the encryption state to keep track of used nonces to avoid repeating and the decryption state is unused.

To reduce $\mathsf{sAEAD}_{\text{QUIC}}$'s level-1 AEAD security to the underlying AEAD's nonce-based AEAD security, we first recall that the nonce-based AEAD security is defined as two separate parts, privacy and authenticity. For privacy, the adversary guesses the secret bit of a left-or-right encryption oracle but cannot make queries with a repeated nonce. The associated advantage is denoted by $\mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{AEAD}}(\mathcal{A})$. For authenticity, the adversary tries to forge a valid ciphertext (together with a nonce and an associated data), given an encryption oracle (without the secret bit). The associated advantage is denoted by $\mathbf{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathsf{AEAD}}(\mathcal{A})$. Now, we are ready to prove the following theorem.

Theorem 14. *For any PPT adversary $\mathcal{A}$, there exist PPT adversaries $\mathcal{B}$ and C such that:*

$$\mathbf{Adv}^{\mathsf{aead\text{-}1}}_{\mathsf{sAEAD}_{QUIC}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathsf{AEAD}}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{AEAD}}(C) \ .$$

Proof. Consider two games G_0 and G_1. G_0 is the real experiment for $\mathcal{A}$ and G_1 is the same as G_0 except that it will always return $\perp$ for decrypt queries. Denote $\Pr_i$ as the advantage of $\mathcal{A}$ in G_i. $|\Pr_0 - \Pr_1|$ is bounded by the probability that $\mathcal{A}$ forges a new valid ciphertext given $b = 1$, which by definition is bounded by $\mathbf{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathsf{AEAD}}(\mathcal{B})$ for some PPT adversary $\mathcal{B}$. Then, note that according to the $\mathsf{sAEAD}_{\text{QUIC}}$ construction nonces in AEAD encryption queries never repeat and G_1 can be simulated by an PPT adversary C against the nonce-based AEAD privacy security, which implies $\Pr_1 \leq \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{AEAD}}(C)$. Therefore, we have $\mathbf{Adv}^{\mathsf{aead\text{-}1}}_{\mathsf{sAEAD}_{\text{QUIC}}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathsf{AEAD}}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{AEAD}}(C)$. □

B.2 TLS 1.3's Stateful AEAD Scheme and its Security

Next, we show TLS 1.3's stateful encryption scheme $\mathsf{sAEAD}_{\text{TLS}}$ constructed from a nonce-based AEAD scheme $\mathsf{AEAD} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as follows:

```
sGen():
    $k_e \xleftarrow{\$} \mathsf{Gen}(), k_m \xleftarrow{\$} \{0,1\}^n$
    $(st_e, st_d) \leftarrow (0, 0)$
    return $(k_e, k_m)$

sEnc($k, ad, m, st_e$):
    $(k_e, k_m) \leftarrow k$
    $c \leftarrow \mathsf{Enc}(k_e, k_m \oplus st_e, ad, m)$
    $st_e \leftarrow st_e + 1$
    return $(c, st_e)$
```

```
$\underline{\mathsf{sDec}(k, ad, ct, st_d)}$:
    if $st_d = \perp$, return $(\perp, \perp)$
    $(k_e, k_m) \leftarrow k$
    $m \leftarrow \mathsf{Dec}(k_e, k_m \oplus st_d, ad, ct)$
    if $m = \perp$,
        $st_d \leftarrow \perp$
    otherwise,
        $st_d \leftarrow st_d + 1$
    return $(m, st_d)$
```

Note that in the above TLS's stateful encryption scheme, nonce repeating is prevented by the increasing counter kept by the encryption state st_e. Following a very similar argument as in the above proof of Theorem 14, one can show that the level-4 AEAD security of $\mathsf{sAEAD}_{\text{TLS}}$ is also reduced to the nonce-based AEAD security of AEAD. This result has been proved by previous work (Theorem 3 in [12]), but their stateful AEAD security definition is slightly different from ours. For instance, in their game the adversary needs to distinguish ciphertexts from random, while in our game the adversary distinguishes ciphertexts of two messages.

APPENDIX C
MSACCE-STD SECURITY OF TLS 1.3 OVER TFO AND QUIC OVER UDP

C.1 TFO+TLS 1.3's msACCE-std Security

Due to the high similarity among the abundant TLS 1.3 proofs in the MSKE model (and its extensions) and a security proof in our msACCE-std model, we show a proof sketch below.

Previous works [103] and [11] respectively proved that the TLS 1.3 draft-16 (EC)DHE full handshake and draft-14 PSK-(EC)DHE 0-RTT resumption handshake are secure in the MSKE model based on the collision resistance of the hash function, unforgeability of the signature and MAC schemes, PRF security of the key derivation function, and pseudorandom function oracle Diffie-Hellman (PRF-ODH) assumption [4, 29, 116]. Their MSKE security, which captures only the key exchange phases, ensures the Bellare-Rogaway-style key secrecy [101] (i.e., the stage keys are indistinguishable from random ones) with various authentication properties (for which our msACCE-std model focuses on the unilateral server authentication). These results derived the overall TLS 1.3 security using a *compositional* approach, i.e., composing a secure key exchange protocol (e.g., the TLS 1.3 handshake protocol) in the MSKE model with an arbitrary secure symmetric key protocol (e.g., the TLS 1.3 record protocol). However, as stated in [11], this generic composition result only works for key-independent, forward-secret, external, and non-replayable stage keys. In particular, it does not apply to the final session keys in full handshakes or the interim handshake keys because they are used internally in the key exchange phases. Besides, it does not apply to the 0-RTT keys, which are replayable and non-forward-secret. In order to adjust their security results to prove TLS 1.3's Server Authentication and level-4 Channel Security in our model, we need to address a few TLS 1.3 updates and model differences as follows.

First, we show that the security results in [103, 11] for old TLS 1.3 drafts can be extended to the standard TLS 1.3 [5], i.e., the standard TLS 1.3 (EC)DHE full handshake and PSK-(EC)DHE 0-RTT handshake are secure in the MSKE model.

1) The multi-stage key generation procedures are updated in the just-released TLS 1.3. Recall that TLS 1.3 performs key derivation in the extract-then-expand paradigm [117] using the HMAC-based Extract-and-Expand Key Derivation Function (HKDF), which consists of two functions $\mathsf{HKDF.Extract}$ and $\mathsf{HKDF.Expand}$. In particular, it first extracts an internal secret (e.g., early secret, handshake secret, and master secret), then expands it (twice) to derive the corresponding stage key. The latest standard TLS 1.3 performs an expand-then-extract procedure instead of a single extract procedure for the extraction of the handshake secret and master secret. However, these two additional expand steps do not affect the MSKE security because they only add a constant (single) query to $\mathsf{HKDF.Expand}$, leading to a larger constant for its PRF advantage. Besides, compared to [103], such extra expand steps help TLS 1.3's MSKE security no longer rely on the PRF security of the underlying HMAC primitive of both HKDF functions.

2) The message flows of the PSK-(EC)DHE 0-RTT resumption handshake are updated in the just-released TLS 1.3. The 0-RTT `Finished` message is replaced by a pre-shared key (PSK) binder. They are both HMAC values generated with very similar procedures and have the same purpose, i.e., to authenticate the `ClientHello` message and to bind the current resumption session with the assoicated full session. Such a replacement does not affect the TLS 1.3's MSKE security. Besides, a new `EndOfEarlyData` message is added as an indicator to end 0-RTT data transmission. This is an empty handshake message independent of key generation so does not affect the security either.

Then, based on the above extended TLS 1.3 MSKE security, we can apply the security results in [10] to get the Multi-Level&Stage security of the combination of the TLS 1.3 full handshake and 0-RTT resumption handshake. Referring to their notions [10], our msACCE-std model focuses only on two modes, i.e., the (EC)DHE full handshake and

PSK-(EC)DHE 0-RTT resumption handshake, and two levels, i.e., one level of full handshakes followed by one level of 0-RTT resumption handshakes.

Finally, we show that the above TLS 1.3 security result in the Multi-Level&Stage model [10] can be augmented to prove TLS 1.3's Server Authentication and level-4 Channel Security.

1) The above security result guarantees server authentication, i.e., a client oracle that has set its final session key must share the same session identifier with a unique partner server oracle. However, their session identifier is defined as *unencrypted* key exchange messages in order to capture key independence (i.e., revealing independent stage keys in the same session does not break the unrevealed stage key's secrecy). We instead use a "real" encrypted session identifier to simplify our model and make reducing KE Payload Integrity to Server Authentication easy. (Note that an unencrypted session identifier may correspond to many valid encrypted session identifiers but KE Payload Integrity requires no modification in the encrypted payload). To prove Server Authentication, we need to follow their proof of the TLS 1.3 Multi-Level&Stage server authentication to replace handshake keys with independent and random values, then use $\mathsf{sAEAD}_{\mathrm{TLS}}$'s AEAD oracles to simulate encrypted key exchange messages in $\mathtt{sid}_{\mathrm{TLS}}$ and the decryption of them. In this way, Server Authentication can be reduced to the TLS 1.3 Multi-Level&Stage server authentication and the AEAD security.

2) To prove level-4 Channel Security, we follow their proof of the TLS 1.3 Multi-Level&Stage security to replace all stage keys with independent and random values and then use the AEAD oracles to simulate encrypted key exchange messages and $\mathsf{Encrypt}, \mathsf{Decrypt}$ queries. In this way, level-4 Channel Security can be reduced to the TLS 1.3 Multi-Level&Stage security and the level-4 AEAD security of $\mathsf{sAEAD}_{\mathrm{TLS}}$. Note that the AEAD oracles are also used to simulate post-handshake messages like `NewSessionTicket`. This bypasses the composition issue [9] faced by the MSKE model (and its extensions), in which the application keys in full handshakes cannot be

composed with secure symmetric key protocols because these keys are used internally in the key exchange phase to encrypt `NewSessionTicket` messages.

C.2 UDP+QUIC's msACCE-std Security

It has been proven in [32] that QUIC is QACCE-secure in the random oracle model based on the unforgeability of the signature scheme, the computational Diffie-Hellman (DH) assumption [118], and the nonce-based AEAD security. Note that msACCE-std with $\mathsf{sAEAD}_{\text{QUIC}}$ is semantically equivalent to QACCE with nonce-based AEAD and $\mathsf{get\_iv}$ (defined in [32]), so their QACCE security results can be trivially adapted to show that UDP+QUIC achieves Server Authentication and level-1 Channel Security in our msACCE-std model. Note that msACCE-std security relies on the level-1 AEAD security of $\mathsf{sAEAD}_{\text{QUIC}}$ instead of the nonce-based AEAD security of the underlying AEAD, but the former can be reduced to the latter as shown in Appendix B.

REFERENCES

[1] E. Rescorla, "The transport layer security (tls) protocol version 1.2," 2008.

[2] ——, "Http over tls," 2000.

[3] Y. Sheffer, R. Holz, and P. Saint-Andre, "Summarizing known attacks on transport layer security (tls) and datagram tls (dtls)," *RFC 7457*, 2015.

[4] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, "On the security of tls-dhe in the standard model," in *Advances in Cryptology–CRYPTO 2012*, Springer, 2012, pp. 273–293.

[5] E. Rescorla, *The transport layer security (tls) protocol version 1.3*, RFC 8446, Aug. 2018.

[6] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, "A cryptographic analysis of the tls 1.3 handshake protocol candidates," in *ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15, New York, NY, USA: ACM, 2015, pp. 1197–1210, ISBN: 978-1-4503-3832-5.

[7] C. Cremers, M. Horvat, S. Scott, and T. v. Merwe, "Automated analysis and verification of tls 1.3: 0-rtt, resumption and delayed authentication," in *2016 IEEE Symposium on Security and Privacy (SP)*, vol. 00, 2016, pp. 470–485.

[8] H. Krawczyk and H. Wee, "The optls protocol and tls 1.3," in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, IEEE, 2016, pp. 81–96.

[9] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, *A cryptographic analysis of the tls 1.3 draft-10 full and pre-shared key handshake protocol*, Cryptology ePrint Archive, Report 2016/081, `https://eprint.iacr.org/2016/081`, 2016.

[10] X. Li, J. Xu, Z. Zhang, D. Feng, and H. Hu, "Multiple handshakes security of tls 1.3 candidates," in *Security and Privacy (SP), 2016 IEEE Symposium on*, IEEE, 2016, pp. 486–505.

[11] M. Fischlin and F. Günther, "Replay attacks on zero round-trip time: The case of the tls 1.3 handshake candidates," in *Security and Privacy, 2017 IEEE European Symposium on*, IEEE, 2017, pp. 60–75.

[12] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, J. Protzenko, A. Rastogi, N. Swamy, S. Z. Béguelin, K. Bhargavan, J. Pan, and J. K. Zinzindohoue, "Implementing and

proving the TLS 1.3 record layer," in *2017 IEEE Symposium on Security and Privacy, SP 2017*, IEEE Computer Society, 2017, pp. 463–482, ISBN: 978-1-5090-5533-3.

[13] K. Bhargavan, B. Blanchet, and N. Kobeissi, "Verified models and reference implementations for the tls 1.3 standard candidate," in *Security and Privacy (SP)*, IEEE, 2017, pp. 483–502.

[14] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe, "A comprehensive symbolic analysis of tls 1.3," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2017, pp. 1773–1788.

[15] J. Brendel, M. Fischlin, and F. Günther, "Breakdown resilience of key exchange protocols: Newhope, tls 1.3, and hybrids," in *European Symposium on Research in Computer Security*, Springer, 2019, pp. 521–541.

[16] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of computer and system sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[17] S. M. Bellovin and M. Merritt, "Encrypted key exchange: Password-based protocols secure against dictionary attacks," in *Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, IEEE, 1992, pp. 72–84.

[18] J. Blocki, M. Blum, A. Datta, and S. Vempala, "Towards human computable passwords," *ArXiv preprint arXiv:1404.0024v4*, 2016.

[19] G. Gebhart, *Tipping the scales on https: 2017 in review*, Dec. 2017.

[20] *HTTPS encryption on the web - Google transparency report*, 2018.

[21] J. Aas, *Let's encrypt: Looking forward to 2019*, Dec. 2018.

[22] G. Linden, *Make data useful*, 2006.

[23] IP Latency Statistics — Verizon Enterprise Solutions, *Verizon enterprise solutions*, 2018.

[24] J. Roskind, "Quic(quick udp internet connections): Multiplexed stream transport over udp," *Technical report, Google*, 2013.

[25] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan, "Tcp fast open," in *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, ACM, 2011, p. 21.

[26] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain, *Tcp fast open*, RFC 7413, RFC, Fremont, CA, USA: RFC Editor, Dec. 2014.

[27] J. Iyengar and M. Thomson, *Quic: A udp-based multiplexed and secure transport*, Nov. 2019.

[28] M. Thomson and S. Turner, *Using transport layer security (tls) to secure quic*, Nov. 2019.

[29] H. Krawczyk, K. G. Paterson, and H. Wee, "On the security of the tls protocol: A systematic analysis," in *Annual Cryptology Conference*, Springer, 2013, pp. 429–448.

[30] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and S. Zanella-Béguelin, "Proving the TLS handshake secure (as it is)," in *Proceedings of CRYPTO*, 2014.

[31] M. Fischlin and F. Günther, "Multi-stage key exchange and the case of google's quic protocol," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2014, pp. 1193–1204.

[32] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru, "How secure and quick is QUIC? provable security and performance analyses," in *Security and Privacy, 2015 IEEE Symposium on*, 2015, pp. 214–231.

[33] *Have i been pwned?* https://haveibeenpwned.com/, Accessed: 2019-12-09.

[34] Verizon, *2019 data breach investigations report*, https://enterprise.verizon.com/resources/reports/2019-data-breach-investigations-report.pdf.

[35] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *2012 IEEE Symposium on Security and Privacy*, IEEE, 2012, pp. 553–567.

[36] FIDO, *Specifications overview*, https://fidoalliance.org/specifications/, Accessed: 2019-12-10.

[37] W. W. W. Consortium *et al.*, *Web authentication: An api for accessing public key credentials level 1 – w3c recommendation*, https://www.w3.org/TR/webauthn, Mar. 2019.

[38] F. Alliance, *Client to authenticator protocol (ctap) – proposed standard*, https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-

`client-to-authenticator-protocol-v2.0-ps-20190130.html`, Jan. 2019.

[39] A. Boldyreva, S. Chen, P.-A. Dupont, and D. Pointcheval, "Human computing for handling strong corruptions in authenticated key exchange," in *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, IEEE, 2017, pp. 159–175.

[40] P.-A. Dupont, "Advanced password-authenticated key exchanges," PhD thesis, 2018.

[41] S. Chen, S. Jero, M. Jagielski, A. Boldyreva, and C. Nita-Rotaru, "Secure communication channel establishment: Tls 1.3 (over tcp fast open) vs. quic," in *European Symposium on Research in Computer Security*, Springer, 2019, pp. 404–426.

[42] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proceedings of the 1st ACM conference on Computer and communications security*, ACM, 1993, pp. 62–73.

[43] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *International conference on the theory and applications of cryptographic techniques*, Springer, 2000, pp. 139–155.

[44] P. Rogaway, M. Bellare, and J. Black, "Ocb: A block-cipher mode of operation for efficient authenticated encryption," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 3, pp. 365–403, 2003.

[45] T. Kohno, A. Palacio, and J. Black, "Building secure cryptographic transforms, or how to encrypt and mac.," 2003.

[46] C. Boyd, B. Hale, S. F. Mjølsnes, and D. Stebila, "From stateless to stateful: Generic authentication and authenticated encryption constructions with application to tls," in *Cryptographers Track at the RSA Conference*, Springer, 2016, pp. 55–71.

[47] M. Bellare, T. Kohno, and C. Namprempre, "Breaking and provably repairing the ssh authenticated encryption scheme: A case study of the encode-then-encrypt-and-mac paradigm," *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, no. 2, pp. 206–241, 2004.

[48] K. G. Paterson, T. Ristenpart, and T. Shrimpton, "Tag size does matter: Attacks and proofs for the tls record protocol," in *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, 2011, pp. 372–389.

[49] M. Abdalla, M. Bellare, and P. Rogaway, "The oracle diffie-hellman assumptions and an analysis of dhies," in *Cryptographers Track at the RSA Conference*, Springer, 2001, pp. 143–158.

[50] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie, "Universally composable password-based key exchange," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2005, pp. 404–421.

[51] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, IEEE, 2001, pp. 136–145.

[52] M. Abdalla, D. Catalano, C. Chevalier, and D. Pointcheval, "Efficient two-party password-based key exchange protocols in the uc framework," in *Cryptographers Track at the RSA Conference*, Springer, 2008, pp. 335–351.

[53] E. Bresson, O. Chevassut, and D. Pointcheval, "Security proofs for an efficient password-based key exchange," in *Proceedings of the 10th ACM conference on Computer and communications security*, ACM, 2003, pp. 241–250.

[54] R. Canetti and T. Rabin, "Universal composition with joint state," in *Annual International Cryptology Conference*, Springer, 2003, pp. 265–281.

[55] T. Matsumoto and H. Imai, "Human identification through insecure channel," in *Workshop on the Theory and Application of of Cryptographic Techniques*, Springer, 1991, pp. 409–421.

[56] C.-H. Wang, T. Hwang, and J.-J. Tsai, "On the matsumoto and imais human identification scheme," in *International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 1995, pp. 382–392.

[57] T. Matsumoto, "Human–computer cryptography: An attempt," *Journal of Computer Security*, vol. 6, no. 3, pp. 129–149, 1998.

[58] X.-Y. Li and S.-H. Teng, "Practical human-machine identification over insecure channels," *Journal of Combinatorial Optimization*, vol. 3, no. 4, pp. 347–361, 1999.

[59] S. Dziembowski, "How to pair with a human," in *International Conference on Security and Cryptography for Networks*, Springer, 2010, pp. 200–218.

[60] A. Juels and S. A. Weis, "Authenticating pervasive devices with human protocols," in *Annual international cryptology conference*, Springer, 2005, pp. 293–308.

[61] J. Bringer and H. Chabanne, "Trusted-hb: A low-cost version of hb+ secure against man-in-the-middle attacks," *IEEE Transactions on Information Theory*, vol. 54, no. 9, pp. 4339–4342, 2008.

[62] J. Bringer, H. Chabanne, and E. Dottax, "Hb++: A lightweight authentication protocol secure against some attacks," in *Second international workshop on security, privacy and trust in pervasive and ubiquitous computing (SecPerU'06)*, IEEE, 2006, pp. 28–33.

[63] J. Munilla and A. Peinado, "Hb-mp: A further step in the hb-family of lightweight authentication protocols," *Computer Networks*, vol. 51, no. 9, pp. 2262–2267, 2007.

[64] X. Leng, K. Mayes, and K. Markantonakis, "Hb-mp+ protocol: An improvement on the hb-mp protocol," in *2008 IEEE international conference on RFID*, IEEE, 2008, pp. 118–124.

[65] K. A. Khoureich, "Hhb: A harder hb+ protocol.," in *SECRYPT*, 2015, pp. 163–169.

[66] ——, "Light-hhb: A new version of hhb with improved session key exchange," *Cryptology ePrint Archive, Report 2015/713*, 2015.

[67] N. J. Hopper and M. Blum, "Secure human identification protocols," in *International conference on the theory and application of cryptology and information security*, Springer, 2001, pp. 52–66.

[68] *Rsa securid hardware tokens*, RSA Security, `https://www.rsa.com/en-us/products-services/identity-access-management/securid/hardware-tokens`.

[69] N. Haller, "The s/key one-time password system," IETF, RFC 1760, Feb. 1995, `http://tools.ietf.org/html/rfc1760`.

[70] N. Haller, C. Metz, P. Nesser, and M. Straw, "A one-time password system," IETF, RFC 2289, Feb. 1998, `http://tools.ietf.org/html/rfc2289`.

[71] M. Nystroem, "The EAP protected one-time password protocol (EAP-POTP)," IETF, RFC 4793, Feb. 2007, `http://tools.ietf.org/html/rfc4793`.

[72] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen, "Hotp: An HMAC-based one-time password algorithm," IETF, RFC 4226, Dec. 2005, `https://tools.ietf.org/html/rfc4226`.

[73] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, "Totp: Time-based one-time password algorithm," IETF, RFC 6238, May 2011, `https://tools.ietf.org/html/rfc6238`.

[74] *Google authenticator*, Google, Inc. `https://support.google.com/accounts/answer/1066447?hl=en&rd=1`.

[75] K. G. Paterson and D. Stebila, “One-time-password-authenticated key exchange,” in *Australasian Conference on Information Security and Privacy*, Springer, 2010, pp. 264–281.

[76] V. Feldman, W. Perkins, and S. Vempala, “On the complexity of random satisfiability problems with planted solutions,” in *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, ACM, 2015, pp. 77–86.

[77] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko, “The one-more-rsa-inversion problems and the security of chaum’s blind signature scheme.,” *Journal of Cryptology*, vol. 16, no. 3, 2003.

[78] J.-S. Coron, J. Patarin, and Y. Seurin, “The random oracle model and the ideal cipher model are equivalent,” in *Annual International Cryptology Conference*, Springer, 2008, pp. 1–20.

[79] T. Holenstein, R. Künzler, and S. Tessaro, “The equivalence of the random oracle model and the ideal cipher model, revisited,” in *Proceedings of the forty-third annual ACM symposium on Theory of computing*, ACM, 2011, pp. 89–98.

[80] Y. Dai and J. Steinberger, “Indifferentiability of 8-round feistel networks,” in *Annual International Cryptology Conference*, Springer, 2016, pp. 95–120.

[81] M. Abdalla and D. Pointcheval, “Simple password-based encrypted key exchange protocols,” in *Cryptographers track at the RSA conference*, Springer, 2005, pp. 191–208.

[82] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson, “TCP congestion control with a misbehaving receiver,” *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 5, 1999.

[83] L. Joncheray, “A simple active attack against TCP,” in *USENIX Security Symposium*, 1995.

[84] R. Abramov and A. Herzberg, “TCP ack storm DoS attacks,” in *IFIP International Information Security Conference*, 2011, pp. 29–40.

[85] Centre for the Protection of National Infrastructure, “Security assessment of the transmission control protocol,” Centre for the Protection of National Infrastructure, Tech. Rep. CPNI Technical Note 3/2009, 2009.

[86] A. Kuzmanovic and E. Knightly, “Low-rate TCP-targeted denial of service attacks and counter strategies,” *IEEE/ACM Transactions on Networking*, vol. 14, no. 4, pp. 683–696, 2006.

[87] V. A. Kumar, P. S. Jayalekshmy, G. K. Patra, and R. P. Thangavelu, “On remote exploitation of TCP sender for low-rate flooding denial-of-service attack,” *IEEE Communications Letters*, vol. 13, no. 1, pp. 46–48, 2009.

[88] S. Jero, H. Lee, and C. Nita-Rotaru, “Leveraging state information for automated attack discovery in transport protocol implementations,” in *IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015.

[89] Z. Qian, Z. M. Mao, and Y. Xie, “Collaborative TCP sequence number inference attack: How to crack sequence number under a second,” in *ACM Conference on Computer and Communications Security*, 2012.

[90] Y. Cao, Z. Qian, Z. Wang, T. Dao, S. V. Krishnamurthy, and L. M. Marvel, “Off-path TCP exploits: Global rate limit considered dangerous,” in *USENIX Security Symposium*, 2016.

[91] Y. Gilad and A. Herzberg, “Off-path attacking the web,” in *WOOT*, 2012, pp. 41–52.

[92] Z. Qian and Z. M. Mao, “Off-path TCP sequence number inference attack - how firewall middleboxes reduce security,” in *IEEE Symposium on Security and Privacy*, 2012, pp. 347–361.

[93] R. Morris, “A weakness in the 4.2 BSD unix TCP/IP software,” AT&T Bell Leboratories, Tech. Rep., 1985.

[94] P. Watson, “Slipping in the window: TCP reset attacks,” CanSecWest, Tech. Rep., 2004.

[95] S. Jero, E. Hoque, D. Choffnes, A. Mislove, and C. Nita-Rotaru, “Automated attack discovery in tcp congestion control using a model-guided approach,” in *Network and Distributed Systems Security Symposium (NDSS)*, 2018.

[96] J. Postel, *Transmission control protocol*, RFC 793 (Standard), 1981.

[97] ——, *User datagram protocol*, RFC 768 (Standard), 1980.

[98] I. Swett, *QUIC deployment experience @Google*, `https://www.ietf.org/proceedings/96/slides/slides-96-quic-3.pdf`, 2016.

[99] M. Fischlin, F. Günther, G. A. Marson, and K. G Paterson, “Data is a stream: Security of stream-based channels,” in *Annual Cryptology Conference*, Springer, 2015, pp. 545–564.

[100] C. Patton and T. Shrimpton, “Partially specified channels: The TLS 1.3 record layer without elision,” in *ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2018.

[101] M. Bellare and P. Rogaway, “Entity authentication and key distribution,” in *Annual International Cryptology Conference*, Springer, 1993, pp. 232–249.

[102] D. A. McGrew and J. Viega, “The security and performance of the galois/counter mode (gcm) of operation,” in *International Conference on Cryptology in India*, Springer, 2004, pp. 343–355.

[103] B. J. Dowling, “Provable security of internet protocols,” PhD thesis, Queensland University of Technology, 2017.

[104] A. Studer and A. Perrig, “The coremelt attack,” in *European Symposium on Research in Computer Security*, 2009, pp. 37–52.

[105] A Langley and W Chang, *Quic crypto*, 2016.

[106] K. Hu and Z. Zhang, “Security analysis of an attractive online authentication standard: Fido uaf protocol,” *China Communications*, vol. 13, no. 12, pp. 189–198, 2016.

[107] C. Panos, S. Malliaros, C. Ntantogian, A. Panou, and C. Xenakis, “A security evaluation of fidos uaf protocol in mobile and embedded devices,” in *International Tyrrhenian Workshop on Digital Communication*, Springer, 2017, pp. 127–142.

[108] O. Pereira, F. Rochet, and C. Wiedling, “Formal analysis of the fido 1. x protocol,” in *International Symposium on Foundations and Practice of Security*, Springer, 2017, pp. 68–82.

[109] C. Jacomme and S. Kremer, “An extensive formal analysis of multi-factor authentication protocols,” in *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, IEEE, 2018, pp. 1–15.

[110] I. B. Guirat and H. Halpin, “Formal verification of the w3c web authentication protocol,” in *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security*, ACM, 2018, p. 6.

[111] R. Cramer and V. Shoup, “Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack,” *SIAM Journal on Computing*, vol. 33, no. 1, pp. 167–226, 2003.

[112] K. Igoe, D. McGrew, and M. Salter, *Fundamental elliptic curve cryptography algorithms*, RFC 6090, Feb. 2011.

[113] K. Moriarty, B. Kaliski, J. Jonsson, and A. Rusch, *Pkcs #1: rsa cryptography specifications version 2.2*, RFC 8017, Nov. 2016.

[114] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A concrete security treatment of symmetric encryption," in *Proceedings 38th Annual Symposium on Foundations of Computer Science*, IEEE, 1997, pp. 394–403.

[115] P. Rogaway, "Authenticated-encryption with associated-data," in *Proceedings of the 9th ACM conference on Computer and communications security*, ACM, 2002, pp. 98–107.

[116] J. Brendel, M. Fischlin, F. Günther, and C. Janson, "Prf-odh: Relations, instantiations, and impossibility results," in *Annual International Cryptology Conference*, Springer, 2017, pp. 651–681.

[117] H. Krawczyk, "Cryptographic extraction and key derivation: The hkdf scheme," in *Annual Cryptology Conference*, Springer, 2010, pp. 631–648.

[118] M. Abdalla, M. Bellare, and P. Rogaway, "The oracle diffie-hellman assumptions and an analysis of dhies," in *Cryptographers Track at the RSA Conference*, Springer, 2001, pp. 143–158.

VITA

Shan Chen is a doctoral student at Georgia Tech. His current research focus is cryptography. Back in his home country China, he earned his master's degree in computer science and bachelor's degree in mathematics. He was born in Weifang, a beautiful city known as the kite capital of the world.