Annals of Telecommunications (2021) 76:187–202
https://doi.org/10.1007/s12243-020-00791-2

Transparency of SIM profiles for the consumer remote SIM provisioning protocol

Abu Shohel Ahmed[1,2] · **Mukesh Thakur**[2,3] · **Santeri Paavolainen**[1,2] · **Tuomas Aura**[1]

Received: 17 February 2020 / Accepted: 8 July 2020 / Published online: 19 August 2020

Abstract

In mobile communication, User Equipment (UE) authenticates a subscriber to a Mobile Network Operator (MNO) using credentials from the MNO specified SIM profile that is securely stored inside the SIM card. Traditionally, a change in a subscriber's SIM profile, such as a change in a subscription, requires replacement of the physical SIM card. To address this shortcoming, the GSM Association (GSMA) has specified the consumer Remote SIM Provisioning (RSP) protocol. The protocol enables remote provisioning of SIM profiles from a server to SIM cards, also known as the embedded Universal Integrated Circuit Card (eUICC). In RSP, any GSMA-certified server is trusted by all eUICCs, and consequently any server can provision SIM profiles to all eUICCs, even those not originating from the MNO associated with the GSMA-certified RSP server. Consequently, an attacker, by compromising a server, can clone a genuine SIM profile and provision it to other eUICCs. To address this security problem, we present SIM Profile Transparency Protocol (SPTP) to detect malicious provisioning of SIM profiles. SPTP assures to the eUICC and the MNO that all SIM provisioning actions—both approved and unapproved—leave a permanent, non-repudiatable trail. We evaluate security guarantees provided by SPTP using a formal model, implement a prototype for SPTP, and evaluate the prototype against a set of practical requirements.

Keywords Consumer RSP · SIM profile cloning · eSIM security · Transparency

1 Introduction

In a mobile network environment, each User Equipment (UE), such as a mobile phone, contains a Subscriber Identity Module (SIM) card, a type of Universal Integrated Circuit Card (UICC). The SIM card stores SIM profiles consisting of the International Mobile Subscriber Identity (IMSI) number and secret keys. These are required for the UE to authenticate itself to the Mobile Network Operator (MNO) with the Authentication and Key Agreement [1, 2] protocol.

Traditionally, each UICC is provisioned once with a SIM profile, and the provisioned UICC is inserted to a UE. However, this one-time provisioning of a SIM profile to a UICC is inflexible as it requires a new UICC to be created and delivered to the customer for each new mobile subscription. To address this problem, the GSM Association (GSMA), a global mobile operator alliance, has specified a consumer remote SIM provisioning (RSP) [3, 4] protocol that can flexibly provision new SIM profiles to an embedded UICC (eUICC). An eUICC is an extension of regular UICC that allows multiple provisioning of SIM profiles to the same eUICC. This approach eliminates the need for a new UICC when a user changes mobile subscriptions.

There are two versions of the RSP protocol. One version targets machine-to-machine type devices, while the other targets consumer-type devices such as mobile phones and wearables. This paper focuses on the consumer version of RSP.

In the consumer RSP protocol, to provision a new subscription, the MNO first orders a SIM profile from a Subscription Manager Data Preparation server (SM-DP+, *server*). At this phase, the server and the MNO share subscriber credentials, e.g., IMSI and secrect keys, that will be provisioned to the UE using a SIM profile (*profile*). Later on, the SIM profile is delivered by the server to the eUICC at the request of the equipment owner.

In the RSP protocol, the GSMA Certificate Authority (CA) and its sub-CAs issue certificates both to the servers

✉ Abu Shohel Ahmed
abu.ahmed@aalto.fi

[1] Aalto University, Espoo, Finland

[2] Ericsson, Kirkkonummi, Finland

[3] University of Helsinki, Helsinki, Finland

and to the eUICCs. Both the servers and the eUICCs trust the GSMA CA, and thus they can form trust relations using the GSMA CA as a trust anchor. Based on the trust relations, any server, upon receiving a request from any eUICCs, can provision SIM profiles to it, which in turn can validate the SIM profile as originating from a GSMA trusted server. Consequently, a third party without GSMA CA certificate is unable to provision SIM profiles to an eUICC, even if they can communicate with it.

Unfortunately, existing SM-DP+ servers can be still targeted, and if anyone of them is compromised by a malicious adversary, the adversary gains the ability to provision cloned profiles to *all* eUICCs. More precisely, an attacker can clone IMSIs and secret keys from honestly provisioned SIM profiles and, later on, remotely provision the same IMSIs and the secret keys to attacker-controlled eUICCs. The attacker can use these cloned SIM profiles for identity theft and billing fraud.

Before the adoption of the RSP, profiles were provisioned to UICCs only in secure off-line facilities. This approach effectively limited the number of UICCs or eUICCs that an attacker could access. In contrast, RSP [4] allows any server to provision profiles to any eUICCs. The increase in the number of servers[1] and the eUICC's blind trust towards the servers increase the attack surface and the probability of attacks from a compromised server towards eUICCs.

The threat of a compromised SM-DP+ server must not be underestimated. As a reference to another widely used technology, Web Public Key Infrastructure (Web PKI) is embedded in modern browsers used in billions of devices. The browsers trust hundreds of X.509 public CAs as the issuers of domain name certificates. There are many examples of CA compromise [6–8] due to inside attacks, software error, operational failure, or government compulsion. It can be seen that a single rogue public CA can compromise the security of the whole Public Key Infrastructure (PKI) by issuing rogue domain name certificates. Similarly, in RSP, a server can be compromised and the rogue server can compromise security for a large number of eUICCs and MNOs by cloning of SIM profiles.

This paper proposes a new security mechanism, SIM Profile Transparency Protocol (SPTP), which provides a verifiable and fast mechanism to detect provisioning of cloned profiles using the consumer RSP protocol. In SPTP, the server registers an audit record for each provisioned profile to an append-only and tamper-proof ledger. The eUICC verifies the presence of the registered audit record prior to accepting a new SIM profile. Therefore, a SIM profile will not be accepted by a device unless the server has published tamper-proof audit records for the profile. Additionally, MNOs monitor the ledger and cross-check provisioned SIM profiles to its own internal records of subscriptions. By significantly increasing the risk of detection and decreasing the time to detection, the monitoring of audit records deters a malicious but cautious attacker from performing the malicious profile provisioning attack on the RSP protocol. SPTP aims to be an extension of the GSMA RSP protocol that minimally impacts both the eUICCs and the servers. SPTP is inspired by a similar solution, Certificate Transparency (CT) [9] for the Web PKI, and related efforts [10–12].

This paper makes the following contributions:

- We present a type of profile cloning attack performed by a compromised server.
- We offer an extension to the consumer RSP protocol, called SPTP, which provides transparency for the provisioned SIM profiles. We formally evaluated security guarantees offered by SPTP to mitigate the profile cloning attack.
- We have implemented an SPTP prototype using an Ethereum-based blockchain [13], and evaluated it against a set of practical requirements. We present implementation guidance and critical analysis of our work.

The rest of this paper is structured as follows. Section 2 discusses related work and Section 3 provides a foundation for important identifiers related to RSP and a summary description for the RSP protocol. In Section 4, we present a type of profile cloning attack. Section 5 introduces SPTP to address the profile cloning attack. Section 6 formally analyzes the security properties of SPTP and Section 7 presents a privacy analysis of the protocol. Section 8 presents a prototype of SPTP and evaluate the prototype against a set of practical requirements. Finally, in Section 9, we critically analyze several actors and alternate design choices for SPTP.

2 Related work

The term *transparency* is generally used to refer to mechanisms that allow others to share a common view of the actions of an entity, and to easily scrutinize changes. Specifically, Certificate Transparency (CT) describes how certificate issuance activity by a CA is made publicly auditable [9]. Transparency mechanisms have been researched in other areas such as in revocation transparency [11], key transparency [14], and application transparency [15]. These address the transparency with the help of authenticated data structures also known as transparency logs. The transparency log appends all changes submitted to it in a cryptographically assured and an auditable manner. The

[1] As of writing this article, the number of servers is 14 [5], and this number is increasing with time.

transparency log allows an untrusted prover to prove the authenticity of a sequence of records to a verifier [16]. The prover provides a proof to the verifier, which can cryptographically verify for the presence of each record. For example, a Merkle hash tree [17] can be used as a log, in which the hash tree's internal nodes store the hashes of their children, and the leaf children store the binding value. The root of the tree is periodically published so that the prover cannot later deny the existence of the previous states.

CT [9] logs every certificate at issuance, thus allowing monitors to check for duplicate entries for the same name from the log, which makes it possible to detect the issuance of a rogue certificate. On the downside, such a system requires active monitoring and gossiping among the verifiers or among designated auditors to detect misuse by the central entity.

CONIKS [14] provides privacy-preserving key transparency by logging the name-to-key binding for each client and only allowing a certain number of entities to access the binding. The client uses bindings from the log as an authoritative source of information when contacting another client. Compared with the CT, in which external monitors check the consistency of bindings in the log, in CONIKS, each client monitors its own bindings. While CONIKS has relatively low bandwidth requirements for clients monitoring their own binding, even this can require excessive resources for constrained or embedded devices, including a UICC that can not securely use the computational capability of a UE.

Bonneau [18] uses the Ethereum blockchain to log commitment for bindings to achieve key transparency. The use of the Ethereum blockchain as a tamper-proof ledger ensures the consistency and append-only property for the stored bindings. The author claims that the use of blockchain for key transparency achieves security properties similar to a centralized authenticated data structure. Additionally, using the decentralized Ethereum network enables the solution to achieve the transparency of records.

Using a similar idea, decentralized naming services on top of a blockchain, the Ethereum Name Service [19] and BlockStack [20] bind names to keys. Once a bind operation is committed to a blockchain, it becomes transparent to all nodes. Thus, a consumer can use bindings from the blockchain as a trusted source of information. On the downside, the consumer of a binding needs to be a so-called full node capable of verifying blocks and included transactions from the blockchain. Block verification is a resource-intensive activity, which limits the use of blockchains for constrained environments.

Several earlier proposals such as Perspective [21] and SSL observatory [22] use a set of notaries to monitor the history and changes of a public key of an entity. The notary attests its viewpoint at a specific time for a specific public key. The same key can be signed by multiple notaries allowing a client to trust the name-to-key binding based on a quorum policy. The observatory-based trust prevents the binding inconsistency attack between a name and a key. On the downside, such a system requires additional queries by clients to notaries to evaluate the observations, which can result in high latency for the binding consistency check.

We have no knowledge of direct prior works applicable for SIM profile transparency. The closest prior works are CT and CONIKS that provides transparency for a certificate and a key respectively.

3 Background

This section provides an overview of the various technologies that will be needed in the rest of the paper.

3.1 Identifiers

The protocols used and developed in this paper make use of various identifiers, which are summarized here:

International Mobile Subscriber Identity (IMSI) [2] identifies an unique subscriber in an MNO network. The IMSIs are globally administered by the International Telecommunication Union (ITU-T), which allocates an IMSI range to each MNO.

Subscriber Identity Module (SIM) profile [23] is a data package consisting of the IMSI and the secret keys, using which the subscriber is authenticated to the mobile network (e.g., UMTS [24]-based mobile network). Each SIM profile is identified by a unique International Circuit Card Identifier (ICCID) [25].

Embedded Universal Integrated Circuit Card (eUICC) [26, 27] is embedded in the user equipment. The eUICC can contain multiple SIM profiles, each of which must have a unique ICCID. Each eUICC is uniquely identified by an eUICC Identifier (EID), which is linked to an eUICC-specific public-private key pair with a certificate issued by the GSMA Certificate Authority (CA) or its sub-CAs.

3.2 Blockchain and smart contracts

A blockchain is a distributed authenticated data structure in which valid records are appended and shared among the participants [28]. The participants validate each record and append it to a chain of records. A consensus protocol defines the ordering of the records. The append-only property of a blockchain provides a distributed consensus of historical events among the participants without relying on

a centralized party. An attacker, depending on the consensus protocol, needs to outpace the honest nodes to create a forked history of records. Thus, a system with a sufficient number or portion of honest nodes can guarantee the finality of the records.

Many blockchains, such as Ethereum, support *smart contracts*, which are fully deterministic programs that are stored and maintained as part of the blockchain itself. All computations performed by a smart contract are executed by all participating blockchain nodes, ensuring that all parties have the same view of the results and the resulting smart contract state. Once a smart contract is deployed, its life cycle and access policy are managed by the contract itself. For example, a party may develop a smart contract that acts as a neutral clearinghouse in a protocol. Other parties may then audit the deployed contract to verify that it follows the protocol correctly and does not contain deviations, such as backdoors, that would benefit a single party. The fully deterministic and flexible execution model makes it possible to use smart contracts in many cases to replace a centralized processing service.

3.3 Consumer remote SIM provisioning protocol

The RSP system consists of four main types of actors:

1. **MNO** provides a mobile network and orders a SIM profile for a subscriber from a server.
2. **eUICC** is part of a UE and it securely stores SIM profiles.
3. **Subscriber** has a subscription agreement with an MNO. The subscriber triggers the UE, he or she has, to download an MNO specified SIM profile from a server.
4. **Server** provisions a SIM profile to the eUICC.

In RSP, the GSMA CA acts as a common trust anchor, issuing certificates to servers and eUICC manufacturers (EUM). Individual eUICC certificates are subsequently issued by the EUM as part of the manufacturing process. Both the eUICC and the server have certificate chains starting from the GSMA CA to establish trust between them. The trust relations across these actors are summarized in Fig. 1. The trust relations and any issued certificates are presumed to occur before the RSP protocol is used. Note that the RSP protocol assumes that servers, eUICCs, MNOs, and the GSMA CA are always honest actors.

The RSP protocol flow is also visible in Fig. 1 (in solid black line). Below, we summarize three steps for the RSP protocol flow.

1. **Profile order:** The MNO orders a new profile from the server (S). This ordering request includes an IMSI and secret keys that are shared between the MNO and the profile. If the eUICC is already known to the MNO, the ordered profile can be directly linked to the eUICC by the server. Otherwise, the generated profile is linked to an activation code instead.
2. **Profile download initialization:** When a subscriber buys a mobile subscription from an MNO, the MNO provides either an eUICC or an activation code to the subscriber. Later on, the subscriber triggers the download of a profile using the UE.
3. **Profile download:** The server delivers a profile to the eUICC based on either the eUICC identity or the presented activation code. In the latter case, the server would also link the profile to the receiver eUICC. The server also notifies the profile orderer MNO about the profile receiver eUICC.

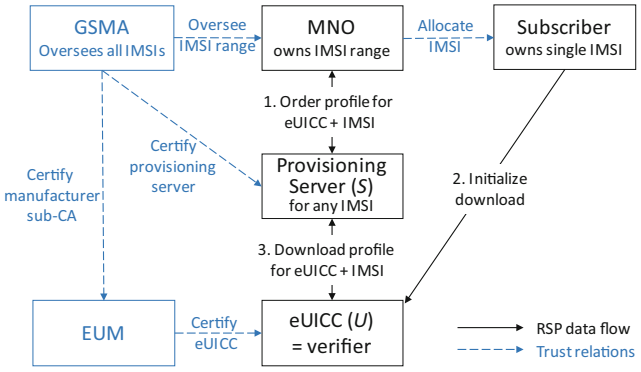

Fig. 1 GSMA consumer remote SIM provisioning and trust relations

In the second step of the protocol, the eUICC may know the server address in several ways. For example, the eUICC can be preconfigured with the server address; the activation code can include the server address; the user can manually enter the server address to the UE; the eUICC receives the server address from a discovery server. The triggering methods are also context-dependent. For example, the user can explicitly trigger a download process using the delivered eUICC; the user can trigger a download by entering the activation code; the UE can automatically discover the availability of a new profile for the eUICC from a default discovery node. Personal mobile devices will probably use one of the first two methods, while for Internet-of-Things (IoT) devices, the latter approach is preferred.

The RSP protocol ensures confidentiality and authenticity in the third step by, first, using a Transport Layer Security (TLS) tunnel with the server where the UE or the eUICC only authenticates the server with certificate chains that have the GSMA CA as the trust anchor. Both the server and the eUICC then mutually authenticate each other using a common handshake [4] protocol with certificate chains that have the GSMA CA as the trust anchor. All protocol messages for the profile download happens over this tunnel.

4 Problem definition

The consumer RSP protocol assumes that all servers with a GSMA certificate are entirely honest because they can provision profiles to any eUICC. This trust assumption is similar to the trust assumption of the Web PKI, in which browsers blindly trust domain name certificates issued by any public CA. There have been cases where malicious actors were able to issue rogue Web certificates for established domains by compromising public CAs including DigiNotar [29] and Comodo [30]. Detecting such compromises takes time due to the lack of public auditability of the certificates. Similarly, in RSP, eUICCs blindly trust any server for provisioning profiles. An attacker can exploit this trust assumption for malicious provisioning of cloned profiles.

A clone profile [31] is a duplicate of an original profile containing the same subscriber credentials, i.e., IMSI and secret keys. In RSP, an attacker in possession of a compromised server can clone an IMSI and secret keys from an earlier MNO-ordered genuine SIM profile and provision the same IMSI and the secret keys to multiple eUICCs. A cloned profile allows the attacker to impersonate itself as a valid subscriber to the MNO using the cloned credentials from those eUICCs. This can be further leveraged for identity theft to other services which use the subscriber's phone number as a trusted identifier (via text message-based two-factor authentication, for example).

Note that it is possible to clone a profile outside of the RSP protocol, e.g., a compromised eUICC or an MNO may leak the profile, or an attacker in possession of physical access to an eUICC may extract the profile. These existing techniques for profile cloning are not affected due to the introduction of the RSP protocol.

We argue that while the RSP protocol is secure against external adversaries without inside access, the whole RSP ecosystem is completely compromised if even a single server gets breached. Upon such a scenario, the attacker uses two main weaknesses of the RSP protocol to perform the attack, as mentioned earlier. These are the following: (1) eUICCs blindly trust any GSMA-certified server for provisioning profiles without any accountability for the server misbehavior and (2) the MNO or and other auditors cannot reliably validate the IMSIs used by a server in the SIM profiles it provisions. We are going to address these two weaknesses of the RSP protocol by using a profile transparency mechanism.

5 SIM profile transparency

We propose a security mechanism, SIM Profile Transparency Protocol (SPTP), to implement transparency for SIM profiles provisioned with the RSP protocol. The protocol defines new participating entities, and new steps that provide accountability of SIM profile delivery to all parties.

5.1 Actors and their requirements

SPTP introduces two new actors, the **Private Index Calculator (PIC)** and the **Transparency ledger (T)** in addition to the existing actors from the vanilla RSP protocol. The new actors are shown in Fig. 2 in green color. The job of the PIC is to calculate a private index for an IMSI and deliver it to authenticated parties. The transparency ledger is an append-only ledger and is used in the protocol to ensure that any SIM profile provisioning action is permanently recorded.

In the following, we state the requirements for the two new actors, and new requirements for existing actors from the vanilla RSP protocol. Please also refer to Table 1 for a summary of different assumptions and requirements places on all of the actors described below.

- PIC calculates a verifiable private index for an IMSI. The index preserves privacy of the IMSI, i.e., anyone only having the index cannot link it to the IMSI. The index is also verifiable, i.e., anyone in possession of both the index and a proof can verify that the index is indeed derived from the IMSI.
- Transparency ledger is an append-only ledger that permanently stores records. It provides an interface to others to read and monitor the consistency of the records it stores.
- Server must register a record in the transparency ledger that binds an IMSI within a profile to the eUICC that downloads the profile.
- MNO must check the legitimacy of the records in the ledger, i.e., it looks for records of profiles for IMSIs within the MNO's IMSI range that have not been ordered by the MNO.
- eUICC checks that a record for the profile it has received is indeed included in the ledger.

Additionally, all parties participating in the transparency ledger consistency protocol must continuously verify that only valid transactions from permissioned entities are included in the ledger.

5.2 Design of PIC

It is important to construct the private index carefully when storing a permanent record to avoid exposing sensitive information. For example, using IMSI, or even a hash of an IMSI can reveal sensitive information such as the changes of a record in the transparency ledger and their times for a particular IMSI of a subscriber. Consequently, the private

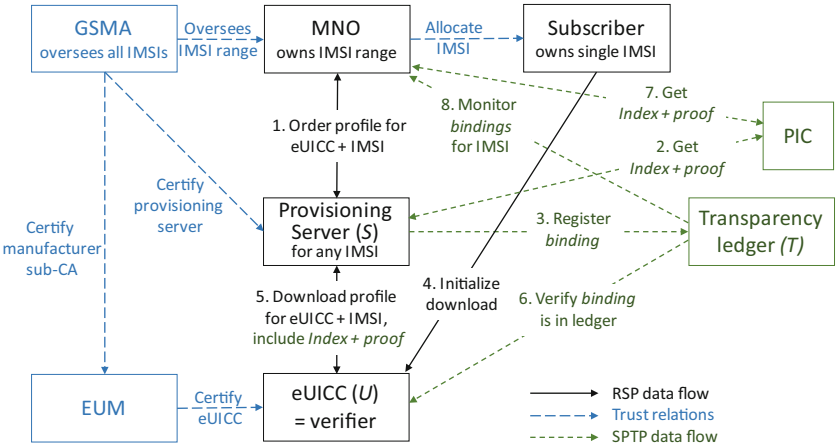

Fig. 2 Message flows in the SIM Profile Transparency Protocol (SPTP)

index should prevent a third party from correlating changes of an IMSI.

To ensure subscriber's privacy, the PIC uses a Verifiable Random Function (VRF) [32, 33] to calculate a private index. It does this by generating a deterministic random output and a publicly verifiable precommitment proof for an input IMSI. The precommitment proof is later revealed to a limited number of authorized parties. A party in possession of both the output and the precommitment proof can cryptographically verify that the output is indeed generated from the same IMSI.

An IMSI cannot be derived by a third party only in possession of the output index. This approach protects the privacy of IMSI by preventing enumeration attacks. Note that VRF always produces deterministic output for an input, i.e., the same input results in the same output. The job of the PIC is to check permissions of anyone who wants to compute the index for an IMSI. In our case, this permission is limited to servers and MNOs. The permission can be further restricted to MNOs such that it can only query for an authorized IMSI range. Note also that eUICCs do not require access to the PIC, they receive the index and the precommitment proof directly from the server. As a safety measure, the PIC may also limit the rate of the index queries for the IMSI range of one MNO and raise the alarm for too aggressive querying.

In the following, we describe how to calculate a private index. Let us assume that the PIC has a key pair (*PK, SK*), where *PK* is the public and *SK* is the private part. Therefore, to calculate deterministic random output *index* and deterministic precommitment proof VrfProof_{IMSI} for an input *IMSI* the PIC generates the precommitment proof by signing a one-way hash of the IMSI:

$$\text{VrfProof}_{IMSI} = \text{sign}_{SK}(\text{hash}(IMSI)) \quad (1)$$

It then generates the *index* by hashing the proof:

$$index = \text{hash}(\text{VrfProof}_{IMSI}) \quad (2)$$

The *index* becomes visible to all parties capable of reading the transparency ledger, while VrfProof_{IMSI} is only released to authorized parties to verify the correctness of the *index*. Any verifier with knowledge of the public key *PK* and VrfProof_{IMSI} can cryptographically verify the correctness of the *index*, i.e., an output *index* is indeed derived from an input *IMSI*.

5.3 Design of transparency ledger

The consistency requirement and the append-only property of the transparency ledger may be accomplished by different methods, such as using a centrally managed, but authenticated and auditable data structure [9], or a decentralized structure such as blockchain [28]. Specifically, we require that the underlying technology of the transparency ledger provides **non-equivocation**, **proof-of-inclusion**, and **state auditability** to meet the consistency and the append-only property. For reference, we describe how these can be accomplished in a decentralized permissioned blockchain below.

A decentralized permissioned blockchain is defined as a blockchain with an admission control policy that defines the entities which are allowed to join the blockchain network, and what operations they are allowed to perform within the network and on the blockchain state. We envision that a blockchain-based transparency ledger allows only parties approved by the GSMA CA, i.e., servers and eUICCs, and approved MNOs to join the network. Furthermore, only servers are allowed to write new records to the ledger. The eUICCs and MNOs are able to read the ledger for auditing purposes, but not modify it. A blockchain-based transparency ledger thus meets the requirements as:

– **Non-equivocation**: a blockchain provides, at a minimum, eventual consistency and consensus of the records. SPTP, as described below, is secure as long as consistency and consensus across all participants are accomplished within some upper bound time limit.

Table 1 Actors and their assumptions

Actor	Assumptions
PIC	The PIC has been set up with a public-private key pair, which is used to authenticate the precommitment proof of the private index. The PIC is trusted by the servers, MNOs, and eUICCs. The PIC is also configured with a policy that defines servers and MNOs allowed to query the private index to determine an IMSI. The PIC is assumed to be honest
Ledger	The ledger is an authenticated data structure, where all participating parties can authenticate and identify the origin of all operations, and the validity of those operations independently. We assume the necessary trust hierarchy, and access control policies regarding the ledger have already been configured to the servers, MNOs, and eUICCs. We also assume different parties monitor the consistency of the ledger
Server	The honest server accepts SIM profile provisioning requests from MNOs, generating and delivering profiles to the eUICC, and recording the provisioned profiles to the transparency ledger. A *malicious server* may deviate from the protocol in any way it wants
eUICC	The eUICC is configured with the GSMA CA hierarchy used to authenticate the provisioned SIM profile. We assume eUICCs are honest in SPTP. We also assume eUICCs are preconfigured with the contact address and a trust anchor for the ledger
MNO	MNOs send SIM profile provisioning requests to servers and authenticate themselves to the ledger and to the PIC with predefined trust relation. We assume MNOs are honest, as they want to protect their business against SIM cloning

- **Proof-of-inclusion**: SPTP requires that all provisioned SIM profiles are recorded in the transparency ledger in a way that can be independently audited as distinct transactions [34]; combined with non-equivocation, this ensures that all proofs-of-inclusions are permanent.
- **State auditability**: in a blockchain, all modifications to the ledger state are accomplished via unique transactions recorded in the append-only log. Since a blockchain consensus requires that all parties agree on the current blockchain state, as replayed from the initial blockchain state, any discrepancy or attempted illegal modification is detected. All and any changes to the transparency ledger state can be attributed to specific transactions, and as all transactions must be signed by the originating party (server), all modifications are attributable to distinct entities.

Furthermore, the use of a decentralized blockchain ensures that the transparency ledger itself is not a single point of failure, and cannot be subjugated by the attack scenario we are assuming, i.e., the compromise of a single server.

The transparency ledger permanently stores a record and provides an interface to verify that the record indeed exists in the ledger. In SPTP, each record consists of an *index* and a *binding* where the *index* is derived by the PIC for an IMSI and the *binding* commits an IMSI of a profile to an eUICC that receives the profile. In the blockchain case, this implies storing the record in persistent storage implemented usually by a form of Merkle Tree. In the Merkle tree, the path to the leaf represents the address to the leaf and a value is inserted for any address with a non-zero value. When an address does not exist, it means the leaf is empty and no value exists. In simple terms, to store a record, the *index* represents the address to which the *binding* is inserted as a leaf to the tree. To verify that a record exists, i.e., the address pointed by the *index* exists and it includes a value, we simply perform the proof-of-inclusion of the transaction that includes the record to the tree and that this transaction is included in the latest published block. Anyone having read access to the ledger can read the *binding* for an *index* (i.e., address) in the tree and can perform proof-of-inclusion check.

5.4 Provisioning a profile in SPTP

The protocol flow is shown in Fig. 2 with the familiar message flows from the RSP protocol in black color and new protocol messages in green color. We describe each step in detail below (see also Table 2 for a summary of all of the notation introduced below).

5.4.1 Step 1: Order profile

The first step in SPTP is identical to the vanilla RSP protocol with the MNO ordering a profile from the server.

5.4.2 Step 2: Retrieve private index and proof

The first deviation from vanilla RSP protocol is that the provisioning server must fetch a private index and proof from the PIC. This communication is authenticated and secured using the GSMA CA trust hierarchy. The request contains the *IMSI* for which the private index needs to be generated, and the reply consists of *index* and VrfProof_{IMSI}.

Fetch private index for IMSI

$PIC \rightarrow S : (index, \text{VrfProof}_{IMSI})$

The server receives both the *index* and the VrfProof_{IMSI}, using which it can validate the signature of VrfProof_{IMSI} and check that *index* is correct.

Table 2 Notations and their description

Symbol	Description
Profile	Profile contains an *IMSI* and keys using which UE authenticates to MNO
PK, *SK*	Public and private key pair of the PIC
index	A VRF *index* for an *IMSI*
VrfProof_{IMSI}	A VRF precommitment proof for an *index*
hash(. . .)	A cryptographically secure hash of its parameters
hash(*IMSI*, *EID*)	Hash that binds the *IMSI* in a profile with an *EID*, the eUICC identity, of an eUICC that receives the profile
counter	A local *counter* for each *index* maintained by the ledger
SID	Server identifier, a value that uniquely identifies the server in the GSMA CA trust hierarchy
binding	Ledger binds an *index* with a hash(*IMSI*, *EID*), a server identifier *SID*, and a *counter*

5.4.3 Step 3: Register binding

Next, the server sends a registration request to the ledger to bind the *index* to the hash of the *IMSI* and eUICC identity that receives the profile (EID).

Register binding for IMSI and EID

$S \rightarrow T : (index, \text{hash}(IMSI, EID))$

The binding registration can immediately progress after the profile has been ordered (step 1) if the server already knows *EID* of the eUICC. Otherwise, this step occurs when the *EID* becomes known when the subscriber uses an activation code on their UE, thereby providing the *EID* to the server. Note that the protocol flow in Fig. 2 assumes that the server already knows the *EID* at the profile order phase.

While this operation can occur asynchronously with later steps without affecting the security of the protocol, it is probably a good idea for the server to wait until the binding has been committed to the transparency ledger so that the eUICC will be able to immediately validate it later.

5.4.4 Step 4: Initialize download

This step is identical to the vanilla RSP protocol, where the subscriber triggers a SIM profile download using a UE from the provisioning server.

5.4.5 Step 5: Download profile

The semantics of the downloaded profile are unchanged from the vanilla RSP protocol, but the exchange adds *index* and VrfProof_{IMSI} to the response.

Deliver profile to eUICC

$S \rightarrow U : (\text{profile}, index, \text{VrfProof}_{IMSI})$

Before eUICC can take the profile into use, it must verify it and ensure that a proper proof-of-inclusion exists in the transparency ledger in the next step.

5.4.6 Step 6: Verify profile and binding

In this step, the eUICC fetches a *binding* from the transparency ledger for the *index*. The binding consists of the hash of combined *IMSI* and *EID*, the server identity, and a transparency counter value.

Retrieve binding for private index

$T \rightarrow U : (\text{hash}(IMSI, EID), SID, counter)$

Based on this information the eUICC can perform all of the following validation steps:

(a) Verify using the VrfProof_{IMSI} that the received *index* is indeed generated from the *IMSI* of the downloaded profile.
(b) Check that *counter* value from the transparency ledger binding is greater than locally stored *counter* from the previous profile, as identified by the *IMSI*. This ensures that the eUICC will not accept a profile which it has seen earlier for the same *IMSI*. (This check is bypassed for fresh profile since there is no previously stored *counter*.)
(c) Check that locally computed hash(*IMSI*, *EID*) based on the *IMSI* from the profile and locally known EID match those from the binding response.
(d) Check that the *SID* from binding matches the remote end identity of the server in step 5.

If all of the checks pass, the eUICC accepts the obtained profile and updates the locally stored *counter* for the *IMSI*.

5.5 Monitoring the ledger

Monitoring is performed continuously by MNO in steps 7 and 8. The MNO learns about the *IMSI* and the *EID* of the eUICC that downloads the profile from the profile order phase. Using this information, the MNO performs the legitimacy of a *binding* using the following protocol: (1) It locally calculates a hash*IMSI*,*EID* for an ordered profile; (2) It obtains an *index* and a proof VrfProof_{IMSI} from the

PIC; (3) It obtains *binding* for the *index* from the ledger; (4) It compares that the *binding* include the locally calculated hash*IMSI*, *EID*. A mismatch between the two indicates an illegitimate *binding* for an *index*.

The MNO has read permission to the ledger and can obtain all or a range of *binding* from the ledger. Using the principles of authenticated data structure [16], it performs a consistency check on the ledger.

6 Security analysis

This section, at first, introduces the basics of formal security analysis. Next, we present a formal analysis of SPTP.

6.1 Introduction to formal analysis

Formal modeling and model checking is a quality assurance method for the analysis and development of security protocols [35]. A formal model includes the message flow of a protocol at an abstract level, and the model checker tests the expected security properties of the protocol. The model checker usually includes an in-built Dolev-Yao [36] type attacker, which the model checker uses against honest participants of the protocol to violate the expected security properties.

6.2 Formal analysis of SPTP

We model the protocol flow of SPTP and its security requirements formally using ProVerif [37] to increase confidence in the protocol. ProVerif is a tool for formal modeling and automated verification of security protocols and their security properties. In the following, first, we describe assumptions, and an implementation note for the model (see Appendix for the model code). We then describe security goals and their results from the formal analysis.

6.2.1 Assumptions

We model four actors, i.e., eUICCs, a server, a transparency ledger, and an MNO, to resemble the participants of SPTP. We model unbounded sessions of each participant using the process construct provided by ProVerif. We do not model the PIC actor due to the operation performed by the PIC is limited and well defined by earlier studies [33], instead, the model assumes that the PIC provides the security guarantees as expected by SPTP. We do not separately model the user and the UE process; the actions of the user and the UE are embedded within the eUICC process. Similarly, the role of the monitor is embedded within the MNO process. We do not model all the message flows of the RSP protocol. We assume that the delivery of the profile from the server to the eUICC is secure, as stated in the RSP specification. Our model focuses on the message flows of SPTP. In our model, the honest server always uses a new IMSI and subscriber key for a profile while the dishonest server can use the same IMSI and subscriber key as many items as it wishes.

The eUICCs, the server, the transparency ledger, and the MNO each have a public and private key pair in our model. The private key of each key pair is known only to the respective entity while the public part is trusted by others. We use the key pair to establish a secure channel between the entities. The secure channel protects both the secrecy and the integrity of the communication. The honest participants never release their respective private keys to the public channel, while a compromised participant can release the key to the public channel. In ProVerif, a common practice is to release the private key of a participant to the public channel to resemble a compromised participant. We use the same technique to compromise the server.

Implementation note ProVerif supports *table* for persistent storage. The *table* supports the insertion of an element and read an element based on a condition. ProVerif returns one possibility from the *table* when several records match a condition. All possible elements from *table* that match the condition are considered during the reasoning phase. Elements in *table* can only be defined and read by ProVerif processes. However, processes cannot delete an element from the *table*. Due to these characteristics, we choose *table* as the storage for the transparency ledger process. Similarly, we use *table* for storing the latest *counter* value of an *index* in the eUICC process.

SPTP requires a *counter* that sequentially increases for each registration of a *binding* for an *index*. However, ProVerif does not support the concept of *counter*. Several earlier models [38] use a *nonce* as an alternative to the *counter*. Our model uses the same technique to simulate the required behavior of the *counter*.

6.2.2 Security goals

To prevent the cloning of profiles SPTP must provide the following two goals: (1) An eUICC accepts a profile from a server if and only if there exists a unique *binding* for the profile in the ledger; (2) An MNO can detect any mis-issuance or clone of a profile.

Goal 1 SPTP requires that the server registers a *binding* for each profile in the transparency ledger, and the eUICC verifies the existence of the *binding* from the transparency ledger for each download of a profile. Formally, if an eUICC accepts a profile with a *binding* consisting of *index*, *imsiEidHash*, and *counter*, then the server has earlier registered the same *binding* to the transparency ledger.

This can be represented as the correspondence between the registered *binding* in the transparency ledger and the obtained *binding* at the eUICC. For this goal, we are interested in an injective form of correspondence. The injective form of correspondence proves that for each *binding* accepted by the eUICC, there is a unique occurrence of the same *binding* registered at the ledger. This guarantees that the server cannot use the same *binding* multiple times to download the same profile to the same eUICC. The server has to register a new *binding* for each profile even if the profile contents remain the same and target the same eUICC. The correspondence for this goal can be expressed as follows (*imsiEidHash* represents hash*IMSI*, *EID*):

```
event(EuiccAcceeptBind(index,imsiEidHash,counter))
==> inj-event(TlBind(index,imsiEidHash,counter))
```

Here, the event EuiccAcceeptBind is executed by the eUICC process after successful verification of the *binding* while the event TlBind is executed by the transparency ledger process at the registration of a *binding*. Consequently, Goal 1 is satisfied if the correspondence property, i.e., for each instance of the event EuiccAcceeptBind, there is a distinct previous instance of the event TlBind, holds for both honest and the dishonest cases.

Goal 2 The MNO learns the *IMSI* and the *EID* for each profile, either at the profile order or the download phase. The MNO also obtains all registered *bindings* from the transparency ledger. Using these two sets of information, the MNO compares a locally calculated hash*IMSI*, *EID* with the hash$IMSI'$, EID' derived from an obtained *binding*. A mismatch between the two indicates a possible clone of a profile. We test this goal by emitting the event MoIMSICloned when the MNO process finds an inequality between the locally calculated hash*IMSI*, *EID* with the value stored in the table of the transparency ledger. We use the *get* construct of ProVerif to perform inequality checks with all entries within the *table*. We use the following reachability query to test that the MNO can detect a clone of a profile.

```
query event(MoIMSICloned(index,imsiEidHash))
```

Consequently, Goal 2 is met if no instance of the event MoIMSICloned is reachable in the honest case (without cloning, i.e., no false positives), and when all paths of dishonest server cloning result in the event being reachable.

6.2.3 Results

In short, our result is that **Goals 1 and 2 are held for both honest and dishonest servers**, i.e., the binding is always observed, and under no condition can a server clone an IMSI without an MNO detecting the clone. These results assure that SPTP achieves the required security properties to detect provisioning of cloned SIM profiles by a malicious server.

Next, to concretely demonstrate how the model shows the security properties, we discuss several mechanisms a dishonest server can attempt to subvert SPTP, and how they are unable to be accomplished in the protocol.

First, let us assume that the ledger contains a *binding* with a hash between the specific IMSI i and a specific eUICC u. The server then clones a profile with IMSI i and target it to an eUICC u'. The server maliciously sends a profile with the IMSI i to an eUICC $u' \neq u$. During the profile download, the eUICC queries the ledger and receives a *binding* for an *index* related to the IMSI i. Since the hash (in hashi, u) within the received *binding* contains u, and the targeted eUICC is u', this will cause *binding* verification failure at the eUICC, and the profile will be rejected. In this case, the eUICC process will not reach to the event EuiccAcceeptBind.

Second, let us assume that the ledger contains a *binding* with a hash between the IMSI i and a specific eUICC u. The server, in this case, clones a profile with the IMSI i and targets the same profile multiple times to the same eUICC u. In SPTP, the eUICC can obtain a *binding* for an *index* related to the IMSI i from the ledger. Additionally, in SPTP, the eUICC locally stores the last known *counter* value after successful verification of a *binding*. The eUICC accepts a new profile for the same *index* if and only if the received *binding* contains a *counter* value that is greater than the previously stored one. This prevents the attacker that has compromised a server, from provisioning the same profile multiple times to an eUICC without registering a new *binding* for each download of the profile. We prove this statement using the injective correspondence property of Goal 1.

Third, a compromised server may clone a profile and register a malicious *binding*, e.g., a *binding* for a cloned IMSI i. The attacker may able to cause the cloned profile to be downloaded to the eUICC after registering such malicious *binding*. However, due to the nature of the transparency ledger, MNOs can obtain all registered *bindings* from the transparency ledger. They can also detect malicious *bindings* not ordered by an MNO. Thus, any attempt by an attacker to register a malicious *binding* to the ledger will be detected by the monitor, and such a registration leaves a permanent audit trace in the ledger.

In summary, the threat model of the RSP protocol assumes that servers, eUICCs, MNOs, and the GSMA CA are always honest actors. With these assumptions, the RSP protocol prevents cloning of profiles, i.e., an honest server will never prepare a cloned profile. We relax the assumption in SPTP with a malicious attacker who can

compromise a server and use it for the attacker's interest. A malicious but cautious attacker is willing to break the rules, i.e., to divert from the tasks given to it in the protocol specification, only when there is small or no possibility that external monitors could detect the malicious activity. In this case, the SPTP can prevent profile cloning attacks. When the server becomes compromised, the server can mount a profile cloning attack irrespectively towards honest eUICCs; however, such an attack leaves permanent detectable traces in the ledger and an MNO is able to detect such an attack. However, SPTP cannot prevent profile cloning if eUICCs are compromised, or the server uses a non-eUICC option to clone a profile. In this case, the attacker can provision profiles to compromised eUICCs without publishing it to the ledger and leaving no trace to detect such an attack.

7 Privacy analysis

In SPTP, a server only sends a hash(*IMSI*, *EID*) for an *index* to the ledger, and no privacy-sensitive information such as the EID or the IMSI is directly leaked to the ledger. The privacy of the IMSI derives from the security of the VRF, in which knowledge of the precommitment proof is needed to reverse an *index* to the plain IMSI. Similarly, the attacker cannot recover the preimage from a hash of the IMSI and the EID. A powerful attacker can, however, recover the preimage from a hash of the IMSI and the EID if either the IMSI or the EID is known to the attacker. The IMSI is up to 15 digits long while the eUICC identifier is 32 digits long. If the attacker knows the EID, it can perform an exhaustive search of over 10^{15} IMSI digits to find the IMSI.[2] This only works when the attacker has other methods to obtain part of the private data, as the hash of both IMSI and EID is computationally infeasible to break even with advanced hash reversing techniques.

A compromised server may learn an IMSI for a particular subscriber and can receive an *index* for the IMSI from the PIC. The server can then track changes for the *index* from the ledger. This can reveal sensitive information for the subscriber such as the time of a *binding* and future changes to that *index*. Additionally, the server, depending on its computational capability, may able to compute the preimage of the EID from the hash*IMSI*, *EID* when the *IMSI* is known. We acknowledge that such types of privacy breaches are possible by a compromised server. However, for this to succeed, the compromised server must learn the IMSI for the particular subscriber through other means.

It may be possible to deduce the time of activity of a particular *index* from the registration of a *binding*. The last known time of contact to an eUICC from the server can be inferred from the previous connection activity. Still, the accurate end of life for a particular eUICC or IMSI cannot be deduced from such traffic analysis. We should note that MNOs can re-use the same IMSI for different subscriptions and subscribers. It is possible that if the server provisions only a few profiles at a time and the attacker observes a person's activity at a mobile phone shop, the attacker could narrow down the range of *indexes* that may be assigned to the person. However, most servers provision a large number of profiles in a short time interval, which makes it harder to identify the exact *index* assigned to the person. One weakness in the scheme is that if an eUICC is replaced, but the subscriber's IMSI remains the same, then the *index* does not change. The hash within the *binding* will change, allowing an external observer to note that the *binding* for the *index* has changed. They are, however, unable to identify whether this is the same subscriber or the IMSI being re-used for another subscriber. Overall, we acknowledge that there are possibilities for traffic analysis, and some correlations could be detected. However, we do not see this information as practically useful to an external adversary.

8 Implementation

SPTP has the following characteristics: (1) a defined set of servers register bindings to the ledger, (2) multiple known untrusted servers can register bindings to the ledger, (3) the MNOs and eUICCs should not extensively rely on a trusted third party (TTP) for auditability of bindings as it opens the door for an attack by the TTP. We evaluated these requirements using the blockchain applicability model [39], which result in to the use of a permissioned blockchain for the transparency ledger.

We implemented transparency ledger as a smart contract written in Solidity, a language that executes on top of the Ethereum virtual machine. The smart contract can be used for (1) registering a *binding* for an *index* and (2) querying the current state of a *binding*. We also implement a simplified version of a server that registers a *binding* to the ledger and an eUICC that checks the presence of the *binding* in the ledger.

We evaluated the prototype from two perspectives: (1) computation and memory requirements at the eUICC to support SPTP and (2) performance evaluation of the smart contract-based transparency ledger. In our prototype, the PIC generates the VrfProof_{IMSI} for an IMSI by signing the hash of the IMSI with the Elliptic Curve Digital Signature Algorithm (ECDSA). Each VrfProof_{IMSI} is 75 bytes long, and each *index* is 32 bytes long. Thus, SPTP increases the existing profile size from the server to the eUICC by approximately 107 bytes. Considering a normal profile is

[2] In practice, MCC and MNC values in IMSI are predictable, leaving 10 digits to MSIN, further reducing the IMSI search complexity.

several kilobytes in size, the additional 107 bytes of data should minimally impact the latency during the profile download.

In SPTP, an eUICC performs an additional request to the ledger to obtain a *binding* for an *index*. Each *binding* message contains four values: *index*, hash*IMSI*, *EID*, *counter* and the server identity. In our implementation, each of these is 32 bytes long. Thus, a *binding* message is 128 bytes long in total. In SPTP, the eUICC additionally stores the last known *counter* for the *index*. This requires 64 bytes of storage for each *index*. In reality, a normal eUICC downloads less than tens of profiles in its lifetime so that the bandwidth and additional storage should not be a practical concern.

From the computational cost perspective, the eUICC verifies an *index* provided by the server and a *binding* for each download of a profile. We list the number of cryptographic computations required to verify both in Table 3. In the Table, the first part refers to the *index* verification cost, while the second part refers to the cost of verifying a *binding*. We observe that the number of cryptographic computations is small and it should minimally impact the eUICC.

Now, we define a couple of practical requirements to evaluate the transparency ledger prototype. According to GSMA Intelligence [40], each year, around two billion SIM profiles are provisioned worldwide. This equals to 63 profiles per second. Thus, the transparency ledger should support high throughput (at least 63 registration of *bindings* per second and a couple of times higher rate of queries). Note that long latency to register a *binding* negatively impacts the user experience when downloading a profile. Thus, we set a limit of 3 s for the latency to register a *binding*. Finally, the cost to register a *binding* should be economically low because high cost disincentivizes honest servers from registering *bindings*. We observe that several permissioned blockchains can meet these practical requirements. In our prototype environment, we choose Quorum [41], an Ethereum-based permissioned blockchain. Our prototype environment is available online [42].

Table 3 Number of cryptographic computations performed by an eUICC to verify an *index* and a *binding*

No	Operation	Description
1	SHA3 Hash	Compute hash(VrfProof_{IMSI}) to compare it with the obtained *index*
1	SHA3 Hash	Compute hash(*IMSI*)
1	ECDSA signature verification	Verify VrfProof_{IMSI} is a signature for the hash(*IMSI*)
1	SHA3 Hash	Compute hash(*IMSI*, *EID*) to compare it with the *binding*

The prototype environment consists of three Virtual Machines (VM), each with 8 VCPUs, 16 GB RAM, and 100 GB of disk space. We use Gauge [43] for characterizing the performance of Quorum. First, we run performance tests for registration and query *binding* operations using Quorum with Raft consensus [44]. The Raft consensus uses a random leader selection process to add blocks to a chain. Second, we run the same tests for Quorum with Istanbul byzantine fault tolerance (IBFT) consensus. IBFT, in contrast to the Raft, verifies a new block proposed by the leader and can work when some participants are dishonest. For both cases, we set Gauge to register (write) *bindings* with the average rate of 80 transactions per second (TPS). Similarly, we set Gauge to query (read) *bindings* with the average rate of 250 TPS. In both cases, we run the tests for 100 thousand *binding* records with 50 ms block time for the Raft consensus and 5 s block time for the IBFT consensus. Figure 3 shows the achieved average latency for the registration and query *binding* operations in both Raft and IBFT. We observe that a query *binding* operation takes less than 0.1 s, while registration of a *binding* (write) takes between 1.0 and 1.7 s. Thus, the achieved latency and throughput meet the earlier set practical requirements.

We have several observations from the transparency ledger prototype: (1) The achieved latency for both consensus algorithms is almost the same and it meets the requirements set for the experiment. (2) Under a high load, e.g., 100 TPS and for more than one million data points, Quorum with IBFT shows instability in the concurrent map iteration and map write. This happens due to a race condition in the IBFT consensus. (3) Under a high load, e.g., 80 TPS and for more than 50 thousand data points, Quorum with Raft shows instability. It happens due to exponential increases of the pending queue and the random leader node

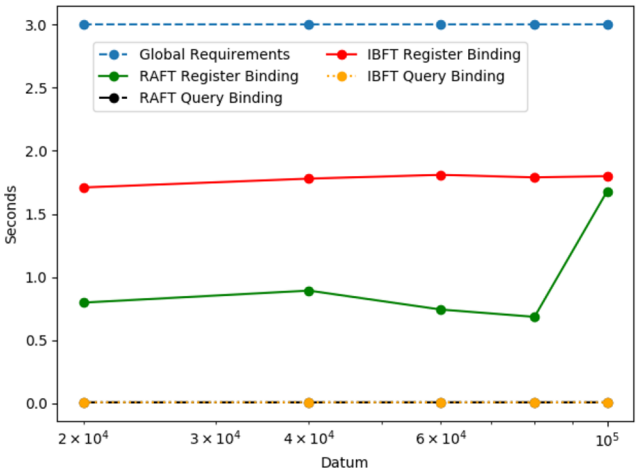

Fig. 3 Average latency for registration and query of bindings

election takes more time resulting in higher latency. The last two observations require further investigation for future improvement of the transparency ledger implementation.

9 Discussion

Remote SIM provisioning can be compared with the web PKI. In RSP, a server has unlimited authority to provision SIM profiles for any IMSI, just like a web CA can issue X.509 certificates for any domain name. In both systems, security concerns arise from this unlimited authority; the trusted entity might not be fully trusted, and its actions need to be monitored. Certificate transparency addresses the problem by making CA actions more transparent. Similarly, SPTP solves the problem by making actions of a profile provisioning server more transparent. The server registers provisioned profiles (mainly the binding between the eUICC and IMSI) into the transparency ledger. When accepting the profile, eUICC requires a binding to the eUICC to be in the ledger. From the ledger, the MNO can monitor the use of its IMSI range. There are some differences between the two systems. The party with the main interest in monitoring SIM provisioning is the MNO, which owns an IMSI range, while in the web PKI, it is the individual web site, which typically has been allocated a single name. Moreover, the web site commonly relies on third-party auditors for the monitoring task. Another difference is that transparency ledger cannot be entirely public. Subscriber privacy requires the use of a permissioned ledger with a formally defined governance model and admission control. Only those authorized for a specific IMSI range can monitor them in the ledger.

There is an alternative solution to SIM profile transparency. Each provisioning could be authorized by the MNO and independently verifiable by the eUICC. That way, the MNO would have the audit data directly, and no ledger would be needed. A shortcoming of this alternative solution is that MNO would be the only auditor. With the transparency ledger, third parties, and even the subscriber can be authorized to audit specific IMSIs, which is our planned future work.

An eUICC with its limited resources is not a suitable platform for directly verifying blocks and data included in the transparency ledger. As an alternative to checking blocks by the eUICC, SPTP depends on a set of trusted full nodes that verify the data and present the requested *binding* to the eUICC. This approach limits the decentralized security for the eUICC.

We could also use notaries to assist eUICCs in verifying *binding* from a transparency ledger. A notary attests a *binding* after observing it in the ledger. This approach ensures that the notaries have seen a *binding*. An eUICC with limited block verification capability can use the attestation as proof-of-inclusion of a *binding* in the ledger. However, this approach has a dependency on trusted notaries and requires further attestations by notaries for each registered *binding*. These requirements complicate the system design. In the future, we plan to work on distributed trust for eUICC. Similarly, we plan to compare several different types of data structures for the transparency ledger.

10 Conclusion

In this paper, we first present a type of profile cloning attack that can be performed using the consumer RSP protocol by an attacker that owns or has compromised a server. Second, we introduce SPTP, a security mechanism for SIM profile transparency, that enables monitors to detect provisioning of cloned profiles to eUICCs by a server. We provide a formal security analysis for SPTP and discuss the effectiveness of it to identify the profile cloning attack. Third, we develop a prototype of SPTP and evaluate our work against a set of practical requirements. We provide practical guidance and future improvement areas for SIM profile transparency.

Funding information Open access funding provided by Aalto University. This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 779984.

Appendix. ProVerif Model

```
(* ===== CHANNELS ===== *)
free c:channel.          (* Public channel *)
(* ===== DATA TYPES ===== *)
type Profile_t.       (* Profile *)
type Host_t.          (* Host address *)
type SK_t.            (* Secret key *)
type PK_t.            (* Public key *)
type Cert_t.          (* Certificate *)
type CertType_t.      (* Certificate type *)
type Nonce_t.         (* Nonce *)
type ImsiEID_t.       (* Commitment *)
```

```
(* ===== CONSTANTS ===== *)
const SK_CI:SK_t [private]. (* GSMA CI Key *)
const uicc,tls,ci:CertType_t.
(* ===== EVENTS ===== *)
event TlBind(bitstring,ImsiEID_t,Nonce_t).
event EuiccAcceeptBind(bitstring,ImsiEID_t,
 Nonce_t).
event MoIMSICloned(bitstring,ImsiEID_t).
(* ===== FUNCTIONS ===== *)
(* Generate public key from private key *)
fun pk(SK_t): PK_t.
(* Sign a certificate with a secret key *)
fun sign_cert(Host_t,PK_t,CertType_t,SK_t):
 Cert_t.
reduc forall H:Host_t,Pk:PK_t,Sk:SK_t,
 C:CertType_t;
    checkcert(sign_cert(H,Pk,C,Sk), pk(Sk))
     = (H,Pk,C).
(* Make a profile using imsi and rest of the
 profile *)
fun make_profile(bitstring,bitstring) :
 Profile_t.
reduc forall imsi:bitstring, P:bitstring;
    exlore_profile(make_profile(imsi,P)) =
     (imsi,P).
(* Make commitment between IMSI and EID *)
fun imsi_eid_commitment(bitstring,Host_t):
 ImsiEID_t.
(* Request certificate *)
letfun certificate(H:Host_t,Pk:PK_t,
 C:CertType_t) =
    sign_cert(H,Pk,C,SK_CI).
(* Create CI certificate *)
letfun ci_cert(Pk:PK_t) =
    new H:Host_t;
    sign_cert(H,Pk,ci,SK_CI).
(* two-way channel *)
fun tochannel(SK_t, PK_t): channel.
equation forall x:SK_t, y:SK_t;
    tochannel(x,pk(y)) = tochannel(y,pk(x)).
(* Storage for the eUICC *)
table uicc_imsi_nonce(bitstring, Nonce_t).
(* Storage for Transparency ledger *)
table bind_sign_storage(bitstring,ImsiEID_t,
 Host_t,Nonce_t).
(** Goal 1 **)
query counter:Nonce_t, imsiEidHash:
 ImsiEID_t,index:bitstring;
   event(EuiccAcceeptBind(index,imsiEidHash,
    counter))
       ==> inj-event(TlBind(index,
        imsiEidHash,counter)).
(** Goal 2 **)
query counter:Nonce_t, imsiEidHash:ImsiEID_t,
 index:bitstring;
   event(MoIMSICloned(index,imsiEidHash)).

(* ===== PROCESSES ===== *)

(* MNO  process *)
let MNO(SK_MNO:SK_t, PK_St:PK_t, PK_TL:PK_t,
 PK_CI:PK_t) =
  !(
   (* === User buys subscription from
    MNO === *)
   in(c, Cert_U:Cert_t);
   let (EID:Host_t, PK_U:PK_t,=uicc) =
       checkcert(Cert_U,PK_CI) in
   new index:bitstring;  (* IMSI and index
    are same in model *)
   (* Message 1: MNO orders a profile to the
    server *)
    let chMNOSP =
      tochannel(SK_MNO,PK_St) in
    out(chMNOSP,(index, Cert_U));
    in(chMNOSP,(=index,=Cert_U));
   (** Message 8: Monitoring the ledger **)
   let imsiEidHash = imsi_eid_commitment
    (index,EID) in
   get bind_sign_storage(=index,=imsiEidHash,
    S1,
      TLnonceInBind1) in
   phase 1;
   get bind_sign_storage(=index,
    imsiEidHashInBind2,S2,
      TLnonceInBind2) suchthat
      imsiEidHash <> imsiEidHashInBind2 in
       (
       event MoIMSICloned(index,imsiEidHash)
       )
       else
    0
  ).

(* Transparency Ledger *)
let TL(SK_TL:SK_t, PK_St:PK_t, S:Host_t) =
  !(
   (** Message 3: Register a binding
    in ledger **)
   let chTPSP = tochannel(SK_TL,PK_St) in
   in(chTPSP,(index:bitstring,imsiEidHash:
    ImsiEID_t));
   new TLnonce:Nonce_t; (* Ledger selected
    counter value *)
   event TlBind(index,imsiEidHash,TLnonce);
   insert bind_sign_storage(index,
    imsiEidHash,S,TLnonce);
   out(chTPSP, (index));
   0
  ).
```

```
(* Server Process *)
let SP(SK_St:SK_t,S:Host_t,PK_TL:PK_t,
 PK_MNO:PK_t,PK_CI:PK_t) =
   !(
    (** Message 1: Profile order request
     from MNO **)
    let chMNOSP =
      tochannel(SK_St,PK_MNO) in
    in(chMNOSP,(index:bitstring, Cert_U:
     Cert_t));
    let (EID:Host_t, PK_U:PK_t,=uicc) =
        checkcert(Cert_U,PK_CI) in
    (* Make profile *)
    new P:bitstring;
    let Profile = make_profile(index, P) in
    (** Message 3: Register a binding **)
    (* Calculate hash of imsi and EID *)
    let imsiEidHash = imsi_eid_commitment
     (index,EID) in
    let chTPSP = tochannel(SK_St,PK_TL) in
    out(chTPSP,(index,imsiEidHash));
    in(chTPSP, (=index));
    (** Message 5: Profile Download **)
    let chUICCSP = tochannel(SK_St,PK_U) in
    in(chUICCSP, Ns:bitstring);
    out(chUICCSP, (Profile, index, Ns));
    (* === Notify MNO for profile download
     success === *)
    out(chMNOSP,(index,Cert_U));
    0
   ) | (
    (* Server process is compromised *)
    out(c, (SK_St));
    0
  ).

(* UE + eUICC Process *)
let UEUICC(PK_CI:PK_t, PK_TL:PK_t, Cert_St:
 Cert_t) =
    new EID:Host_t;new SK_U:SK_t; let PK_U =
     pk(SK_U) in
    let Cert_U = certificate(EID,PK_U,uicc)
     in
    out(c, (Cert_U));
   !(
    (** Message 5: Profile Download **)
    let (S:Host_t,PK_St:PK_t,=tls) =
        checkcert(Cert_St,PK_CI) in
    let chUICCSP = tochannel(SK_U,PK_St) in
    (* Simplified profile download *)
    new Ns: bitstring;
    out(chUICCSP, Ns);
    in(chUICCSP, (Profile:Profile_t,index:
     bitstring,
        It:bitstring, =Ns));
    (** Message 6: Verification of a binding
     **)
    let (=index,P:bitstring) =
     exlore_profile(Profile) in
    let imsiEidHash = imsi_eid_commitment
     (index,EID) in
    get bind_sign_storage(=index,
     =imsiEidHash, =S,TLnonce1) in
    get uicc_imsi_nonce(=index,=TLnonce1) in
      0 (* eUICC rejects profile *)
    else
     insert uicc_imsi_nonce(index,TLnonce1);
     event EuiccAcceeptBind(index,
       imsiEidHash,TLnonce1);
      0
   ).

process
    (* GSMA CI certificate *)
    let PK_CI = pk(SK_CI) in
    let Cert_CI = ci_cert(PK_CI) in
    new SK_TL:SK_t; let PK_TL = pk(SK_TL) in
    new SK_MNO:SK_t; let PK_MNO = pk(SK_MNO) in
    (* Server certificate issued by the CI*)
    new S:Host_t; new SK_St:SK_t; let PK_St
     = pk(SK_St) in
    let Cert_St = certificate(S,PK_St,tls)
     in
    out(c, (Cert_CI, PK_TL, PK_MNO, Cert_St)
     );
    (
      MNO(SK_MNO, PK_St, PK_TL, PK_CI) |
      TL(SK_TL, PK_St, S)  |
      SP(SK_St, S, PK_TL, PK_MNO, PK_CI) |
      !UEUICC(PK_CI, PK_TL, Cert_St)
    )
```

References

1. Holtmanns S, Niemi V, Ginzboorg P, Laitinen P, Asokan N (2008) Cellular authentication for mobile and internet services. John Wiley & Sons, New York
2. 3GPP. 3GPP TS 33.401 Technical specification group services and system aspects 3GPP system architecture evolution (SAE) security architecture, (2018)
3. GSMA. SGP.21 Architecture Specification v2.2, (2017)
4. GSMA. SGP.22 Technical Specification v2.2, (2017)
5. GSMA. SAS Accredited Sites List. https://www.gsma.com/security/sas-accredited-sites/,(2020). Accessed at: 06-01-2020
6. Satapathy A, Livingston J (2016) A comprehensive survey on SSL/TLS and their vulnerabilities. Int J Comput Appl 153(5):31–38
7. Roosa SB, Schultze S (2013) Trust darknet: control and compromise in the internet's certificate authority model. IEEE Internet Comput 17(3):18–25

8. Microsoft. Security Advisory 3046310: improperly issued digital certificates could allow spoofing. https://docs.microsoft.com/en-us/security-updates/securityadvisories/2015/3046310, (2015). Accessed at: 09-01-2020
9. Laurie B, Langley A, Kasper E (2013) Certificate transparency. RFC, 6962
10. Kim TH-J, Huang L-S, Perrig A, Jackson C, Gligor V (2013) Accountable key infrastructure (AKI) a proposal for a public-key validation infrastructure. In: Proceedings of the 22nd International Conference on World Wide Web, pp 679–690
11. Ryan MD (2014) Enhanced certificate transparency and end-to-end encrypted mail. In: NDSS
12. Laurie B, Kasper E (2012) Revocation transparency. Google Research, September, 33
13. Wood G, et al. (2014) Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper 151:1–32
14. Melara MS, Blankstein A, Bonneau J, Felten EW, Freedman MJ (2015) CONIKS: bringing key transparency to end users. In: USENIX Security Symposium, vol 2015, pp 383–398
15. Fahl S, Dechand S, Perl H, Fischer F, Smrcek J, Smith M (2014) Hey, NSA: stay away from my market! future proofing app markets against powerful attackers. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM, pp 1143–1155
16. Miller A, Hicks M, Katz J, Shi E (2014) Authenticated data structures, generically. In: ACM SIGPLAN Notices, vol 49. ACM, pp 411–423
17. Merkle RC (1988) A digital signature based on a conventional encryption function. In: Advances in Cryptology-CRYPTO'87. Springer, pp 369–378
18. Bonneau J (2016) EthIKS: using Ethereum to audit a CONIKS key transparency log. In: International Conference on Financial Cryptography and Data Security. Springer, pp 95–105
19. Ethereum Foundation. Ethereum Name Service. https://ens.domains/, (2020)
20. Ali M, Nelson JC, Shea R, Freedman MJ (2016) Blockstack: a global naming and storage system secured by blockchains. In: USENIX Annual Technical Conference, pp 181–194
21. Wendlandt D, Andersen DG, Perrig A (2008) Perspectives: improving SSH-style host authentication with multi-path probing. In: USENIX Annual Technical Conference, vol 8, pp 321–334
22. Eckersley P, Burns J (2010) The EFF SSL observatory. Internet: https://www.eff.org/observatory
23. Trusted Connectivity Alliance. eUICC Profile Package: Interoperable Format Technical Specification. https://trustedconnectivityalliance.org/wp-content/uploads/2020/01/SIMalliance-eUICC-Profile-Package-Interoperable-Format-Technical-Specification-v2.3.1-and-ASN-Module.zip, (2020)
24. Kaaranen H, Ahtiainen A, Laitinen L, Naghian S, Niemi V (2005) UMTS networks: architecture, mobility and services. John Wiley & Sons, New York
25. ITU (2013) List of issuer identifier numbers for the international telecommunication charge card (in accordance with recommendation ITU-t e.118 (05/2006)). http://www.itu.int/pub/t-SP-OB.1040-2013
26. ETSI. TR 102 216 Smart Cards; Vocabulary for Smart Card Platform specifications. (2003)
27. GSMA. SGP.02 v3.2 Remote Provisioning Architecture for Embedded UICC Technical Specification
28. Crosby M, Pattanayak P, Verma S, Kalyanaraman V (2016) Blockchain technology: beyond bitcoin. Appl Innov 2:6–10
29. Schoen S, Galperin E, Eckersley P (2011) A post mortem on the Iranian DigiNotar attack. https://www.eff.org/deeplinks/2011/09/post-mortem-iranian-diginotar-attack. Accessed: 2019-12-01
30. Richmond R (2011) Comodo fraud incident. https://www.comodo.com/comodo-fraud-incident-2011-03-23.html. Accessed: 2019-12-01
31. Goldberg I (1999) GSM Cloning. http://www.isaac.cs.berkeley.edu/isaac/gsm-faq.html. Accessed at: 09-01-2020
32. Micali S, Rabin M, Vadhan S (1999) Verifiable random functions. In: 40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039). IEEE, pp 120–130
33. Dodis Y (2003) Efficient construction of (distributed) verifiable random functions. In: International Workshop on Public Key Cryptography. Springer, pp 1–17
34. Kiayias A, Miller A, Zindros D (2017) Non-interactive proofs of proof-of-work. IACR Cryptol ePrint Arch 2017(963):1–42
35. Lowe G (1996) Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In: International Workshop on Tools and Algorithms for the Construction and Analysis of Systems. Springer, pp 147–166
36. Cervesato I (2001) The Dolev-Yao intruder is the most powerful attacker. In: 16th Annual Symposium on Logic in Computer Science—LICS, vol 1
37. Blanchet B (2005) Proverif automatic cryptographic protocol verifier user manual
38. Blanchet B (2017) Symbolic and computational mechanized verification of the arinc823 avionic protocols. In: 2017 IEEE 30th Computer Security Foundations Symposium (CSF). IEEE, pp 68–82
39. Wüst K, Gervais A (2018) Do you need a blockchain? In: 2018 Crypto Valley Conference on Blockchain Technology (CVCBT). IEEE, pp 45–54
40. GSMA (2019) GSMA Intelligence - gross additions. https://www.gsmaintelligence.com/metrics
41. Quorum. https://www.goquorum.com/. (2019) Accessed: 2020-01-01
42. SPTP: implementation and formal model. https://bitbucket.org/shohel/transparency/. (2019) Accessed: 2019-12-03
43. Baliga A, Kamat P, Subhod I, Chatterjee S (2018) Performance evaluation of the Quorum blockchain platform. Technical report
44. Ongaro D, Ousterhout JK (2014) In search of an understandable consensus algorithm. In: USENIX Annual Technical Conference, pp 305–319