Threshold Moderation for End-to-End Encrypted Messaging

Max Christman
max@cs.unc.edu
University of North Carolina at Chapel Hill
Chapel Hill, North Carolina, USA

ABSTRACT

Encrypted messaging is used by billions of people daily to ensure private communications. This privacy enables malicious uses, including spam and other abusive or illegal content. Meta's Messenger service uses a system called message franking to moderate end-to-end encrypted (E2EE) messages. This system gives Meta total authority in moderation decisions as well as full responsibility for moderating many millions of messages a day.

This paper modifies message franking to allow many users to act as moderators, while maintaining the system's original security properties. The threshold moderation system uses small groups of moderators responsible for each message, who vote on how to act on each report. The protocol uses threshold secret sharing to condition the undeniability of message reports on a majority vote of the moderators. This decentralized protocol prevents the platform from enforcing unpopular moderation policies while alleviating the burden of moderating every message themselves.

1 INTRODUCTION

After the 2013 revelations that the NSA had been collaborating with tech companies to perform widespread warrantless domestic surveillance, many users had to reevaluate the privacy of their internet communications. In the PRISM surveillance program [4], user data was harvested en masse from corporate servers, often without reference to a legitimate foreign adversary. As much, users became increasingly wary of sharing their data directly with service providers.

This distrust motivated the development of a messaging paradigm under which direct government surveillance was impossible, which came in the form of end-to-end encrypted (E2EE) messaging. In E2EE messaging schemes, messages are encrypted such that the content is only visible to the sender and their intended recipient. This is achieved through ensuring that only the sender and the recipient have the encryption keys, preventing the platform from reading or modifying the messages that it relays.

Numerous E2EE messaging services operate today, allowing billions of people to communicate privately on a daily basis. While this privacy allows users to resist government and corporate surveillance programs, it also makes abuse significantly easier. With E2EE, service providers cannot simply decrypt messages to determine if they are illegal or otherwise objectionable. Instead, the onus is on the recipient to report harmful messages. While this seems to work, most messaging services to do not provide a way to prove that a user sent you a message, in other words, they have deniability.

For the Secret Conversations feature in their Messenger service, Meta designed a system called message franking [1] to solve this problem. When a user sends a message, they are required to send the platform a value (called a commitment) which commits them to the message they are sending. The platform then signs this commitment so that it can later verify its authenticity. If a user comes to the platform with a message, they also have to provide an authentic commitment, which ties the accused user to the message in question. If the commitment verifies, then the platform is convinced that the report is genuine, so long as the commitment scheme and signature scheme are secure.

This system provides a measure of accountability to E2EE messaging without sacrificing confidentiality. However, as the platform is the only party that verifies commitments, they are the sole authority in moderation decisions. This is undesirable for a number of reasons, both practical and ethical. If the platform serves many users, they may have to hire a large team to serve moderation needs. More concerningly, if the platform operates in a country with a repressive government, they may be forced to implement an unpopular moderation policy.

To combat both undermoderation and censorship, we introduce a threshold moderation system. In our system, users are still required to commit to the messages they send. However, these commitments are jointly signed by a group of moderators, using threshold secret shares of a signing key. Upon reading a message report, moderators then share their key shares with the platform, which can then combine them to reconstruct the signing key. With the consensus of a majority of the moderators, the platform is able to verify the commitment and be convinced that the report is legitimate.

This system could enable ordinary users to moderate E2EE messaging platforms, allowing decisions to be made by popular consensus, not by arbitrary corporate policy. Provided reasonable qualifications are required to be a moderator, this system would ensure decisions are made in the interest of the community. Threshold moderation provides a democratic approach to the practice, using the size and wisdom of the community to ensure a safer platform for all.

2 BACKGROUND

2.1 End-to-end encryption

End-to-end encryption, commonly referred to as E2EE, is the practice of only sharing encryption keys between the parties communicating, rather than also sharing them with the platform facilitating the communication. This practice has a number of security benefits, including preventing the platform from reading or tampering with messages, assuming the use of an authenticated encryption scheme. E2EE is widely deployed today in services such as Signal, WhatsApp, and iMessage.

2.2 Message franking

Most E2EE schemes do not provide a way of proving that a given user sent a given message, giving them deniability. While deniable E2EE messaging provides excellent privacy, it also allows users to

send harmful messages with impunity. To solve this problem, in 2016, Meta introduced a protocol called message franking to the Secret Conversations [1] mode of their Messenger platform.

In message franking, when Alice wants to send a message m to Bob, she generates a random nonce n and computes a franking tag T_F as follows:

$$T_F = \text{HMAC}\,(n, m||n)$$

Alice sends the franking tag T_F, the encryption c of the message and randomness, and the conversation context to the platform, who computes the reporting tag R_F under its reporting key K_R as follows:

$$R_F = \text{HMAC}\,(K_R, T_F||\text{context})$$

The platform sends c, T_F, and R_F to Bob, who decrypts c to get m and n. Bob recomputes the franking tag and checks that it agrees with T_F, discarding the message otherwise. When the platform receives a message report, consisting of the message, randomness, conversation context, and reporting tag R_F , they are able to confirm its authenticity as follows:

- The platform recomputes the franking tag from the message and randomness.
- The platform checks if the franking tag and the conversation context verify under the reporting tag, under their MAC key.

2.2.1 Security properties. Facebook claims the following security properties for their message franking system:

- **Confidentiality:** No computationally efficient adversary learns anything about m or n from T_F.
- **Authenticity:** No computationally efficient adversary can produce a m, n, conversation context, and R_F which verifies under K_R, unless the platform computes and publishes:

  $$R_F = \text{HMAC}(K_R, \text{HMAC}(n, m||n)||\text{context})$$

- **Third-party deniability:** Even if R_F was computed as before, no adversary without K_R can produce a m, n, and conversation context such that the above holds.

2.2.2 Shortcomings of message franking. While message franking allows for effective message moderation in E2EE systems, it introduces several problems, particularly when used on a large scale (as it is for Meta's Messenger).

In the original message franking scheme, the platform is the sole moderating party. As such, it is their responsibility to review all message reports and make the corresponding moderation decisions. For a platform with billions of monthly active users, this would require a dedicated moderation team to make sure every message report can be viewed and dealt with. A modified message franking scheme that distributes the burden of moderation across many, potentially independent moderators would be preferable, especially for platforms that lack the resources to hire a dedicated team.

Another consequence of centralizing moderation authority is that it forces the platform to decide on a moderation policy. If the platform is the only one reviewing message reports, this could force them to take a side on political or otherwise contentious issues. This is also undesirable for users, who would prefer that the platform not enforce arbitrary or opaque moderation policies. Both the platform and users would benefit from a system that distributes the authority of moderation across many independent parties.

3 SYSTEM OVERVIEW

3.1 Notation

3.1.1 Message authentication codes (MACs).

- $\text{HMAC}(k, m) \to t$ is a secure many-time hash-based MAC
- $\text{PolyMAC}(k, m) \to t$ is a secure one-time multilinear MAC

More details on the multilinear MAC are given in Appendix 1.1.

3.1.2 Ciphers. $\text{AuthEnc}(k, m) \to c$ and $\text{AuthDec}(k, c) \to m$ or $\perp$ will respectively refer to the encryption and decryption functions which together provide authenticated encryption. $\text{PKEnc}(k, m) \to c$ and $\text{PKDec}(k, c) \to m$ will respectively refer to public key encryption and decryption algorithms that provide semantic security against a chosen-ciphertext attack (IND-CCA).

3.1.3 Interactive protocols. As in [6], we use the following notation to represent an interactive protocol between Alice and Bob:

$$(O_A, O_B) \leftarrow \langle A(I_A), B(I_B)\rangle(I_P)$$

In the above, I_A and I_B are the private inputs for Alice and Bob, O_A and O_B are their private outputs. I_P is the public input for both parties. This notation is also extended to interactive protocols between many parties.

3.2 Threshold moderation system

This paper modifies message franking to distribute the moderation authority, allowing the platform to crowdsource moderation while preventing them from applying unpopular moderation policies. This could be done through enlisting users of the platform as volunteer moderators. These users would then be deemed eligible for membership in moderation pools, i.e. groups of moderators who are responsible for each message.

When Alice sends a message, moderators in her pool generate shares of a MAC key and collectively compute a reporting tag under the sum of these shares. When a message is reported, the moderators can reconstruct the reporting tag and check if it verifies. In this scheme, messages now have deniability for all parties except coalitions of more than the threshold number of moderators.

Once a message report is received, moderators vote by sending their MAC key share to the platform. If enough of them vote in the affirmative, the platform can aggregate these shares to reconstruct the MAC key, recompute the reporting tag, and check that it verifies. If this verifies, the platform is convinced that the message report is valid, and they can take action.

3.3 System definition

A threshold moderation scheme is defined with respect to the following parameters:

- $\mathcal{M}$, the set of all possible moderators
- n, the number of moderators per moderation pool
- t, the threshold value for each moderation pool

Given the above parameters, a *threshold moderation scheme* is defined as follows:

- **SendMessage** is an interactive protocol between Alice and a group of moderators, represented by the following pair of interactive algorithms:

$$(\perp, (c, C, o_i, t_i, \mu_s, \mu_r)) \leftarrow \langle A(sk), \{M_i\}\rangle$$

  Before the protocol, Alice has access to a shared secret key sk. After the protocol, all moderators have a ciphertext c, a commitment C, a sender identifier μ_s, and a recipient identifier μ_r. Additionally, each moderator i has access to an opening share o_i and a MAC share t_i.
- **ReceiveMessage** is an interactive protocol between Bob and a group of moderators, represented by the following pair of interactive algorithms:

$$((m, n, \text{context}, C, \{e_i\}) \text{ or } \perp, \perp) \leftarrow \langle B(sk), \{M_i(k_i)\}\rangle$$

  Before the protocol, Bob has access to a shared secret key sk, while each moderator i has access to a personal symmetric key k_i. After the protocol, Bob has a message m, nonce n, conversation context, commitment C, and encrypted openings $\{e_i\}$.
- **ReportMessage** is an interactive protocol between Bob and a group of moderators, represented by the following pair of interactive algorithms:

$$(\perp, o_i) \leftarrow \langle B(m, n, \text{context}, C, \{e_i\}), \{M_i(k_i)\}\rangle$$

  Before the protocol, Bob has access to a message m, nonce n, conversation context, commitment C, and encrypted openings $\{e_i\}$, while each moderator i has access to a personal symmetric key k_i. After the protocol, each moderator each has access to an opening o_i or nothing.
- **VerifyReport** is an interactive protocol between a group of moderators and the platform, represented by the following pair of interactive algorithms:

$$(\perp, o \text{ or } \perp) \leftarrow \langle \{M_i(o_i)\}, P(k_{priv})\rangle(k_{pub})$$

  Before the protocol, each moderator i has access to an opening o_i, the platform has access to a private key k_{priv}, while both parties have access to a public key k_{pub}. After the protocol, the platform either has an opening o or nothing.

3.4 Design goals

3.4.1 Deployment scenarios. The threshold moderation system is optimal for either (or both) of the following scenarios:

(1) An E2EE platform is unable to employ or recruit enough moderators to effectively handle all message reports.
(2) An E2EE platform faces pressure to apply a particular moderation policy, either internally or externally, e.g. from an oppressive government.

The protocol allows for an arbitrary number of moderators, each placed in many (likely intersecting) moderation pools. As such, it is effective in handling the large volume of message reports inevitably generated from a large platform. Additionally, allowing open participation in moderation makes it highly difficult to enforce an unpopular moderation policy globally.

3.4.2 Functionality goals. As most E2EE messaging systems are low-latency, a threshold moderation system system should not add significant latency on top of the standard message franking protocol. Additionally, such a system should strike a balance between fast turnaround times on message reports and reasonable workloads for participating moderators.

3.4.3 Security goals. The threshold moderation system intends to retain the confidentiality and authenticity of standard message franking, as well as a stronger notion of deniability enabled by threshold reporting tags. The following security goals define security for the threshold moderation system:

(1) **Confidentiality:** If not reported, only the sender and intended recipient learn anything about the contents of the message.
(2) **Authenticity:** If the moderators verify a message report, they must have previously MACed it.
(3) **Threshold deniability:** If fewer than t moderators vote for a message report, the platform should not be able to verify an associated reporting tag. If at least t moderators vote for a message report, the platform should be able to verify if and only if it is legitimate, i.e. if it was in fact MACed by the moderators.

3.4.4 Security assumptions. Only under the following assumptions does the threshold moderation scheme meet the aforementioned security goals:

(1) Moderators follow the protocol, although all may try to learn more than is desired.
(2) HMAC is a secure commitment scheme, meaning that it has computational binding and computational hiding.
(3) PolyMAC is a secure MAC scheme, meaning that it has existential unforgeability against a chosen message attack.
(4) (AuthEnc, AuthDec) provide authenticated encryption, meaning that they have indistinguishability under a chosen plaintext attack (IND-CPA) as well as ciphertext integrity (INT-CTXT).

Note that the first assumption leaves a number of scenarios on the table. For example, it is fine for the platform to save all franking tags, reporting tags, and conversation contexts it receives. Users are also allowed to modify message reports as they please, but this is defended by the platform keeping track of reporting tags.

Importantly, moderators are allowed to disclose the franking tags, reporting tags, MAC keys, and even message plaintext from message reports to any party. If fewer than a threshold of moderators in a moderation pool do this, they will fail to convince anyone of the validity of their claims.

4 PROTOCOL SETUP

4.1 Initializing the platform

The platform generates an asymmetric keypair k_{pub} and k_{priv}, as well as a MAC key k_m.

4.2 Initializing moderators

Users interested in being moderators indicate this desire to the platform, which records their public key in a list of all moderators $\mathcal{M}$. Moderators also each generate a personal many-time key k_i.

4.3 Generating moderation pools

Note: in this section, i indexes moderation pools within the set of all moderation pools whereas j indexes moderators within a moderation pool.

4.3.1 Platform initialization. To generate a pool, the platform:

(1) Chooses one public key uniformly at random from $\mathcal{T}$, the set of highly-trusted moderators. This user is designated as the *head moderator*.
(2) Chooses $n-1$ public keys uniformly at random from $\mathcal{M}$.
(3) The platform sends each of these n public keys to the other $n-1$ public keys, as well as a unique pool identifier μ_i.

4.4 Storing moderation pools

Once all keys are generated, the platform stores the moderation pool with the following representation:

(1) Unique identifier μ_i
(2) Pool parameters n_i and t_i
(3) A pool tag $p_i = \text{HMAC}(k_m, \text{Serialize}((\mu_i, n_i, t_i)))$

Each moderator additionally stores these values in their representation.

The platform can keep a queue of pre-generated moderation pools, ordered by their generation time. They can opt to make this a priority queue as determined by some metric on the pool.

4.5 Public parameters

For the following interactive protocol definitions, all parties are working in the group $\mathbb{Z}_p$, which is the integers modulo a large prime p.

5 THRESHOLD MODERATION SYSTEM

In the following section, we give a decentralized moderation protocol that uses threshold secret sharing to distribute trust.

5.1 SendMessage

When Alice wants to send Bob a message m, they establish a shared secret key sk and participate in the following interactive protocol:

(1) Alice computes ciphertext c as follows:
$$c = \text{AuthEnc}(sk, m||n)$$
(2) Alice generates a nonce n uniformly at random and computes a franking tag as:
$$T_F = \text{HMAC}(n, m||n)$$
(3) Alice sends ciphertext c, franking tag T_F, her sender ID μ_s, and the recipient ID μ_r to the moderators.
(4) The moderators save the timestamp τ, the sender ID μ_s, and the recipient ID μ_r as the conversation context (τ, μ_s, μ_r).
(5) The moderators each generate a threshold share of a MAC key mk_i.
(6) The moderators each compute a share r_i of the reporting tag:
$$r_i = \text{PolyMAC}(mk_i, T_F||\text{context})$$
(7) The moderators aggregate $r = \sum_i r_i$ and save c, T_F, r, and the conversation context for use in ReceiveMessage.

5.2 ReceiveMessage

Immediately after SendMessage finishes, Alice's moderators engage in the following interactive protocol with the user corresponding to μ_r (henceforth referred to as Bob):

(1) The moderators each compute encryptions e_i of their MAC keys:
$$e_i = \text{AuthEnc}(k_i, mk_i)$$
(2) The moderators each send the ciphertext c, franking tag T_F, reporting tag r, encrypted MAC key shares $\{e_i\}$, and the conversation context to Bob.
(3) Bob decrypts the ciphertext to get the message m and nonce n:
$$m||n = \text{AuthDec}(sk, c)$$
(4) Bob checks if $T_F = \text{HMAC}(n, m||n)$, dropping out of the protocol if it does not match.
(5) Bob saves the message m, nonce n, conversation context, reporting tag r, and encrypted MAC key shares $\{e_i\}$ in case he needs to make a report.

5.3 ReportMessage

If Bob feels that the message is abusive, illegal, or otherwise against platform rules, he sends the message m, nonce n, conversation context, reporting tag r, and encrypted MAC key shares $\{e_i\}$ to Alice's moderators, who then perform the following interactive protocol with the platform:

(1) The moderators read the message and context and decide if the message is illegal or otherwise violates platform rules. Each moderator who feels so decrypts their MAC key share as follows:
$$mk_i = \text{AuthDec}(k_i, e_i)$$
Each moderator who feels otherwise drops out of the protocol.
(2) Each remaining moderator sends their MAC key mk_i to the platform.
(3) If they received at least t MAC key shares, the platform aggregates them into a MAC key mk. Otherwise, they discard the threshold MAC key shares and drop out of the protocol.

5.4 VerifyReport

If the platform receives enough MAC key shares to reconstruct a MAC key, they participate in the following interactive protocol with message recipient Bob:

(1) Bob sends the platform the message m, nonce n, conversation context, and reporting tag r.
(2) The platform recomputes the franking tag T_F as follows:
$$T_F = \text{HMAC}(n, m||n)$$

(3) The platform recomputes the reporting tag as follows:

$$\hat{r} = \text{PolyMAC}(mk, T_F||\text{context})$$

(4) If $\hat{r} = r$, the platform is convinced the message report is legitimate, otherwise they are not convinced.

If the platform successfully verifies the report, they can then proceed with the necessary legal and/or disciplinary action.

6 SECURITY

6.1 Confidentiality

If a message is not reported, neither the moderators nor the platform will learn anything about the message. First, consider that the ciphertext c and franking tag T_F together can be viewed as a compactly commiting authenticated encryption [5] to a message m.

By the confidentiality of a ccAE scheme, and thus the hiding of the included commitment, computationally efficient moderators will learn nothing about m from seeing c or T_F. However, this property will clearly not hold if a message is reported.

6.2 Authenticity

Assume for the sake of contradiction that Bob can generate a false message report. This means Bob can produce a message m, nonce n, conversation context, reporting tag r, and encrypted MAC keys e_i such that the reporting tag verifies under the MAC keys, but Alice never sent message m with the given context.

First, assume that the moderators indeed MACed the franking tag and context which Bob provides. For this to be a false report, Bob would have had to find a message m' and nonce n' such that:

$$\text{HMAC}(n', m'||n') = \text{HMAC}(n, m||n)$$

where $m' \neq m$. However, as HMAC has computational binding, such an (m', n') should be impossible for any efficient adversary to compute.

Thus, if Bob can produce an m, n, context, r, and $\{e_i\}$ that verifies, he will have produced a franking tag and context which verify under the moderator's MAC keys, despite them never MACing it. However, as PolyMAC has existential unforgeability, this should be impossible for any efficient adversary. Thus, Bob is not able to generate a false message report that verifies.

6.3 Threshold deniability

First, assume that fewer than t moderators vote for a message report. Without loss of generality, we assume that the platform has $t-1$ threshold shares of mk. Assume that Bob sends the platform a message report, with message m, nonce n, context, and reporting tag r.

Furthermore, assume that the platform can produce inputs mk', $t||c$ to PolyMAC such that $r = \text{PolyMAC}(mk', t||c)$. By the existential unforgeability of the one-time multilinear MAC, we must have that $mk' = mk$. However, as they had only $t-1$ shares of mk, the information-theoretic security of threshold secret sharing implies that they should know nothing about mk. Thus, it is impossible for the platform to produce mk.

Next, assume that at least t moderators vote for a message report. This implies that the platform has (at least) t threshold shares of mk. If Bob sends them a message report, they also have m, n, c, and r as before.

Here, the platform can perform polynomial interpolation on mk_i to get mk. Assume they then compute $\hat{t} = \text{HMAC}(n, m||n)$ and $\hat{r} = \text{PolyMAC}(mk, \hat{t}||c)$. By authenticity, this report should verify if and only if it is genuine.

7 CHOOSING PARAMETERS

Every implementation of the threshold moderation system is defined with respect to a moderation pool size n, and a threshold value t. Platforms are free to set these based on the number available moderators and the level of consensus they expect for moderation decisions.

7.1 Moderation pool metaparameters

Additionally, the platform should set the following parameters which correspond to the generation of moderation pools for sent messages. Platforms wishing to implement threshold moderation should arrive at estimates of the expected number of messages sent per second and number of CPU threads to dedicate to pool generation, denoting the former as ρ and the latter as χ.

From these two estimates, the platform can derive the batch size β as a function of χ and the period τ as a function of ρ. β determines how many pools are generated at once while τ is the time between batches.

7.2 Choosing pool size and threshold

Through careful choice of moderation pool size, the platform can leverage the democratic nature of threshold moderation and achieve representation of diverse viewpoints in moderation pools.

Each moderation pool is parameterized by two values: the number of moderators per pool n and a threshold value $t < n$. These values should be the same for every moderation pool in an instance of the threshold moderation system. The values of these parameters should be determined as a function of two parameters and two estimates:

(1) α : Any given binary opinion with this frequency or greater will be represented in a given moderation pool with likelihood ϕ.
(2) ϕ : The likelihood that views of frequency α or greater will be represented in a given moderation pool.
(3) ψ : The expected likelihood of a moderator responding to any given message report.
(4) $|\mathcal{M}|$: The expected number of moderators.

The lower that the platform chooses α, the more they will ensure non-majority opinions are represented in moderation pools. It is suggested for this value to be lower than 0.5, so as to incorporate some minority opinions.

ψ can be thought of as the platform's tolerance of the occasional moderation pool which doesn't meet their α standard. This value is suggested to be less than 0.01.

7.2.1 Combinatorial method for n. Once these parameters are chosen and estimates made, n can be computed with the following combinatorial method. If α is the portion of people in $\mathcal{M}$ holding a certain binary opinion, we would like to determine the probability

of choosing a pool of size n at random which contains at least one person with each opinion.

Performing the combinatorial calculation, we arrive at the following probability:

$$p = 1 - \frac{\prod_{i=0}^{n-1}\left(\frac{|\mathcal{M}|\cdot\alpha - i}{|\mathcal{M}| - i}\right) + \prod_{i=0}^{n-1}\left(\frac{|\mathcal{M}|\cdot(1-\alpha) - i}{|\mathcal{M}| - i}\right)}{n!}$$

We then take the smallest value t which satisfies the following:

$$p\,(t) = 1 - \frac{\prod_{i=0}^{t-1}\left(\frac{|\mathcal{M}|\cdot\alpha - i}{|\mathcal{M}| - i}\right) + \prod_{i=0}^{t-1}\left(\frac{|\mathcal{M}|\cdot(1-\alpha) - i}{|\mathcal{M}| - i}\right)}{t!} > \phi$$

This t should be used as the threshold value for our scheme. The total pool size n can be computed from the estimate ϕ of the proportion of message reports to which a given moderator will respond as follows:

$$n = \frac{t}{\psi}$$

Now that t and n have been determined, each subsequent moderation pool should be generated with this same threshold value and pool size.

7.3 Reviewing message reports

Given a shared secret key, the moderators can discuss the message report. It is important that such a discussion occur, especially if the platform has chosen an $\alpha < 0.5$. This discussion provides an opportunity for minority voices to express their concerns and potentially convince other moderators to change their mind.

8 FUTURE WORK

While the threshold moderation system improves user privacy by distributing message deniability across several moderators, it is not resistant to actively malicious moderators. In future work, we hope to develop a threshold protocol that uses proofs on threshold secret-shared data to hide the franking tag and context from moderators. Preliminary exploration suggests that the SNIP techniques from [3] can be adapted to the threshold setting without significant difficulty.

A CRYPTOGRAPHIC PRIMITIVES

To understand the foundation on which the threshold moderation system is built, this section gives an overview of some of the underlying cryptographic protocols and primitives, along with the associated notions of security.

A.1 Message authentication codes

A message authentication code or MAC scheme is a triple of algorithms:

- KeyGen$(1^\lambda) \to k$: takes the security parameter λ and returns a key of λ bits.
- Sign$(k, m) \to t$: takes a key k and a message m and returns a MAC t.
- Verify$(k, m, t) \to$ Accept or reject: takes a key k, a message m, a MAC t, and either accepts or rejects.

A MAC scheme is considered secure if it is **existentially unforgeable** under a chosen message attack. Consider the following security game:

- The challenger picks a key uniformly at random from the key space.
- The adversary provides a message m, and the challenger computes $t = \text{HMAC}(k, m)$, returning it to the challenger. They also add (m, t) to their set S of seen message-tag pairs.
- The adversary repeats the previous step as many times as they desire.
- The adversary wins the game iff they can produce a message-tag pair (m', t') such that $\text{HMAC}(k, m') = t'$ and $(m', t') \notin S$.

A MAC scheme is existentially unforgeable under a chosen message attack if there exists no computationally efficient attacker who wins this game with non-negligible probability.

A.1.1 Multilinear MAC. For computing reporting tags, we will instead use a multilinear MAC. First, consider a finite field $\mathbb{F}$, often the integers modulo a large prime p ($\mathbb{Z}_p$). We assume without loss of generality that an l-block message can be represented as as l elements from $\mathbb{F}$.

Sign(m) is defined as follows:

- Choose l elements $k_1, \ldots, k_l$ uniformly at random from $\mathbb{F}$, making k the one-time key.
- Compute the MAC as $t = \sum_i k_i \cdot m_i$ where $\cdot$ is the operation in $\mathbb{F}$.

Verify(k, m, t) is defined as follows:

- Compute $\hat{t} = \sum_i k_i \cdot m_i$.
- If $\hat{t} = t$, output accept, otherwise reject.

This MAC is of interest as it can be easily represented as an arithmetic circuit. In particular, this MAC has one multiplication gate for each block of the message. The results of the multiplications are then summed to get the final MAC value. Thus, if we view the message and key as inputs to the circuit, we can use SNIPs to learn the MAC without knowing the message or the key. We will refer to this MAC (its signing algorithm) as PolyMAC(k, m).

This multilinear MAC scheme is existentially unforgeable when used as a one-time MAC. Consider the security game for existential unforgeability:

- The challenger picks a key uniformly at random from $\mathbb{F}^l$.
- As this is a one-time MAC, the adversary gets only one MAC query. They then submit a (m, t) to the challenger.
- The adversary wins the security game iff $\text{MAC}(k, m) = t$.

However, consider what is implied if an adversary can produce such a (m, t). For this to verify, it must be the case that $\text{MAC}(k, m) = t$, which implies that $\sum_i k_i \cdot m_i = t$, or equivalently that $t - \sum_i k_i \cdot m_i = 0$. If we define $f(m_1, \ldots, m_l) = t - \sum_i k_i \cdot m_i$, we can see that finding a (m, t) that verifies is equivalent to finding a root of $f(m)$. Here, m is a multilinear polynomial of degree one, so by the Schwartz-Zippel lemma [7], the probability of a randomly chosen $r \in \mathbb{F}^l$ being a root of f is at most $1/||\mathbb{F}||$. Thus, an adequately large finite field can provide arbitrary levels of existential unforgeability for this MAC scheme.

A.2 Authenticated encryption

An authenticated encryption scheme is a triple of algorithms:

- KeyGen(1^λ) $\rightarrow k$: takes the security parameter λ and returns a key of λ bits.
- AuthEnc(k, m) $\rightarrow (c, t)$: takes a key k, a message m, and generates a ciphertext c and a MAC t.
- AuthDec(k, c, t) $\rightarrow m$ or $\perp$: takes a key k, a ciphertext c, and a MAC t, and returns either a message m or $\perp$, meaning invalid.

For a scheme to have **authenticated encryption**, it must provide both **semantic security** from a chosen-plaintext attack and **ciphertext integrity** [2].

Indistinguishability under a chosen-plaintext attack (IND-CPA) is achieved when no efficient adversary has non-negligible advantage in the following security game:

- The challenger randomly chooses a key k from the key space, as well as a random bit b.
- The adversary provides a pair of messages (m_0, m_1) to the challenger, who then gives back c as the encryption of the message corresponding to bit b.
- The adversary can repeat this step with as many message pairs as they want.
- The adversary then outputs a bit b', indicating its guess of the true bit b.
- The advantage of the adversary is defined as how much better than random chance they are at guessing the bit b.

Ciphertext integrity (INT-CTXT) is achieved when no efficient adversary can win the following security game with non-negligible probability:

- The challenger randomly chooses a key k from the key space.
- The adversary provides a message m and the challenger returns the encryption c of m under key k. They additionally keep a record of having seen c.
- The adversary can make as many encryption queries as they want.
- The adversary then submits a ciphertext c, which the challenger decrypts.
- If the ciphertext successfully decrypts (doesn't return $\perp$) and the challenger hasn't seen c before, the adversary wins the game.

We will refer to a generic authenticated encryption scheme by the function signatures given above, with one modification. We will combine the ciphertext and MAC together for simplicity. Thus, the ciphertext will contain the message information while simultaneously allowing for authenticity. The function signatures are given as follows:

$$\text{AuthEnc}(k, m) \rightarrow c$$

$$\text{AuthDec}(k, c) \rightarrow m \text{ or } \perp$$

A.3 Threshold secret sharing

Consider a secret that Alice wants to share with n parties. With threshold secret sharing [8], Alice can require that at least t of the parties discuss with each other to recover the secret. The procedure is as follows:

(1) Alice represents the secret as an element $s \in \mathbb{Z}_p$, where p is a large prime.
(2) Alice chooses $t-1$ values uniformly at random from $\mathbb{Z}_p$, calling them $a_1, \ldots, a_{t-1}$.
(3) She then constructs the $t-1$ degree polynomial $q(x)$ as follows:
$$q(x) = s + \sum_{i=1}^{t-1} a_i \cdot x^i$$
(4) Alice computes n "shares" of her secret as $q(i)$ for $i \in \{1, \ldots, n\}$, distributing $(i, q(i))$ to the i-th party.

With this scheme, if t of the parties convence and exchange shares, they can reconstruct the coefficients of $q(x)$, and thus the secret, using polynomial interpolation. If fewer than t parties cooperate, there is a still a uniform prior on the coefficients, making this scheme secure even for a computationally unbounded adversary. This scheme is particularly useful for a setting where Alice expects some, but not all parties to be malicious.

A.4 Secret-shared non-interactive proofs (SNIPs)

Prio [3] introduces the idea of secret-shared non-interactive proofs for the computation of aggregate statistics. In particular, they use SNIPs to prove that individual data values are well formed before aggregating them.

In this scenario, there is a single prover and multiple verifiers. All parties have access to a validation function Valid(x), the verifiers hold additive (not threshold) secret shares of an input x, while the prover has the entire x. The goal of the prover is to convince the verifiers that Valid(x) = 0, where x is value reconstructed from all of the verifiers' shares.

The authors give three security properties for a SNIP:

- **Completeness:** If all verifiers are honest, an honest prover should convince them that Valid(x) = 0.
- **Soundness:** If all verifiers are honest and Valid(x) $\neq$ 0, each verifier will reject x with overwhelming probability.
- **Zero knowledge:** If the prover and at least one verifier are honest, then the verifiers will learn nothing about x other than that Valid(x) = 0.

Prio describes a SNIP protocol which satisifes completeness, soundness, and zero knowledge, the details of which are given in the paper [3].

REFERENCES

[1] 2016. *Messenger Secret Conversations Technical Whitepaper.* Technical Report. Facebook. Accessed: 2023-03-22.

[2] Mihir Bellare and Chanathip Namprempre. 2008. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. *Journal of Cryptology* 21, 4 (July 2008), 469–491. https://doi.org/10.1007/s00145-008-9026-x

[3] Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17).* USENIX Association, Boston, MA, 259–282. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs

[4] Barton Gellman and Laura Poitras. 2013. U.S., British intelligence mining data from nine U.S. internet companies in Broad secret program. https://www.washingtonpost.com/investigations/us-intelligence-mining-data-from-

nine-us-internet-companies-in-broad-secret-program/2013/06/06/3a0c0da8-cebf-11e2-8845-d970ccb04497_story.html

[5] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. 2017. Message Franking via Committing Authenticated Encryption. In *Advances in Cryptology – CRYPTO 2017*. Springer International Publishing, 66–97. https://doi.org/10.1007/978-3-319-63697-9_3

[6] Charlotte Peale, Saba Eskandarian, and Dan Boneh. 2021. Secure Complaint-Enabled Source-Tracking for Encrypted Messaging. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. ACM. https://doi.org/10.1145/3460120.3484539

[7] J. T. Schwartz. 1980. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *Journal of the ACM* 27, 4 (Oct. 1980), 701–717. https://doi.org/10.1145/322217.322225

[8] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (Nov. 1979), 612–613. https://doi.org/10.1145/359168.359176