IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 0, No. 0, pp. 0–0. DOI:10.46586/tosc.v0.i0.0-0

Subverting Telegram's End-to-End Encryption

Benoît Cogliati, Jordan Ethan and Ashwin Jha

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
benoit.cogliati@gmail.com,{jordan.ethan,ashwin.jha}@cispa.de

Abstract. Telegram is a popular secure messaging service with third biggest user base as of 2021. In this paper, we analyze the security of Telegram's end-to-end encryption (E2EE) protocol in presence of mass-surveillance. Specifically, we show that Telegram's E2EE protocol is susceptible to fairly efficient algorithm substitution attacks. While official Telegram clients should be protected against this type of attack due their open-source nature and reproducible builds, this could potentially lead to a very efficient state sponsored surveillance of private communications over Telegram, either on individuals through a targeted attack or massively through some compromised third-party clients. We provide an efficient algorithm substitution attack against MTProto2.0 — the underlying authenticated encryption scheme — that recovers significant amount of encryption key material with a very high probability with few queries and fairly low latency. This could potentially lead to a very efficient state sponsored surveillance of private communications over Telegram, either through a targeted attack or a compromised third-party app. Our attack exploits MTProto2.0's degree of freedom in choosing the random padding length and padding value. Accordingly, we strongly recommend that Telegram should revise MTProto2.0's padding methodology. In particular, we show that a minor change in the padding description of MTProto2.0 makes it subversion-resistant in most of the practical scenarios. As a side-effect, we generalize the underlying mode of operation in MTProto2.0, as MTProto-G, and show that this generalization is a multi-user secure deterministic authenticated encryption scheme.

Keywords: Telegram · MTProto · algorithm substitution · key recovery

1 Introduction

Over the past two decades, smartphones have received widespread adoption across the world. This resulted in a plethora of secure messaging services, such as WhatsApp, Signal, Facebook Messenger, and Telegram, mushrooming over online app repositories.

Telegram, in particular, has more than 500 million active users worldwide [NIF21]. It recently saw a huge surge in subscribers [NIF21] after public outrage against the new privacy policy changes announced by Facebook and WhatsApp. Indeed, over 100 million new users joined Telegram in January 2021 [The21d], making it the most downloaded app across iOS and Android [Cha21].

Telegram offers two conversation modes, the cloud chat mode and the secret chat mode. Messages in cloud chats employ client-server/server-client encryption, and are stored on the Telegram server in encrypted form. So, all messages can be read by the server, allowing for chat history accessibility across devices. Messages in the secret chat mode employ

*** The authors would like to thank all the anonymous reviewers who reviewed and provided valuable comments on this paper. The authors would also like to thank Antoine Joux, Cas Cremers and Thorsten Holz for their help in preparing the final version of this paper. This work was carried out under the framework of the French-German-Center for Cybersecurity, a collaboration of CISPA and LORIA. ***

client-client or end-to-end encryption for only two parties. In this mode, the messages are sent through the server, but can only be decrypted by the two parties involved in the communication.

1.1 MTProto and Its Security

Telegram opted to use a home-brewed original protocol known as MTProto [The21c], both for cloud chats as well as secret chats. At the heart of this protocol lies its eponymous encryption scheme MTProto. In their online technical FAQ [The21b], the Telegram team justified the use of an in-house encryption scheme, as opposed to some well-studied and provably secure encryption scheme, as follows:

> *In order to achieve reliability on weak mobile connections as well as speed when dealing with large files,* MTProto *uses an original approach.*

However, the general cryptographic community is still skeptical of Telegram's security claims and justifications. Indeed, their skepticism is not entirely unfounded, as demonstrated by the attacks on MTProto1.0 [JO16] by Jakobsen and Orlandi. In response to the attacks in [JO16], the Telegram team revised the encryption scheme to MTProto2.0. In [The21a], the Telegram team claims that the latest version of MTProto achieves IND-CCA [BN00] security. However, to the best of our knowledge, a formal proof of security was noticeably missing up until quite recently [AMPS22]. In fact, the Telegram team goes on to say that security notions like IND-CCA while convenient for theoretical and scientific inquiry, do not directly relate to the actual security of communication [The21a], and nothing short of a full plaintext recovery or corruption is practically useful. In our opinion, this limited security policy is quite detrimental to the privacy interests of Telegram's users. Indeed, cryptographic literature is filled with examples [BL16, SBK+17], where theoretical attacks formed the basis for more efficient and practically relevant attacks.

In a quite recent work, Albrecht et al. finally presented a formal IND-CCA security proof [AMPS22] for MTProto2.0. However, within the proof, they make several non-standard assumptions on the underlying building blocks. To a large extent these assumptions are necessitated by the design choices made in MTProto2.0. In addition, Albrecht et al. also propose four attacks on MTProto2.0 by exploiting some vulnerable behaviors exhibited by Telegram clients and servers in some boundary cases. In response, the Telegram team updated the protocol to mitigate these boundary conditions.

1.2 Subversion Attacks

The veiled use of mass surveillance and web traffic interception by government agencies became apparent due to the Snowden revelations. Among other things, it revealed that the government agencies do not just apply intensive cryptanalytic techniques, but also subvert cryptosystems to overcome well established cryptographic hard problems. One such mechanism for subversion is to manipulate the algorithms used in implementations by injecting a backdoor into otherwise secure implementations. A formal treatment of such mechanisms predates the Snowden revelations, and was initiated in a line of work by Young and Yung that they named *kleptography* [YY96, YY97]. Basically, Young and Yung considered an adversary who designs a subverted cryptographic algorithm whose outputs are computationally indistinguishable from the outputs of an unmodified algorithm. The subverted algorithm should leak the secret key through the output, which was achieved using principles similar to Simmons' *subliminal channels* [Sim83].

The Snowden revelations reignited interest in this kind of subversion attacks, starting with the so-called *Algorithm Substitution Attacks* (ASAs) by Bellare et al. [BPR14] against randomized encryption schemes. Their attack relies on influencing the randomness generated in the course of encryption. Specifically, the attack applies to a sub-class of

randomized schemes satisfying a property they call *coin-injectivity*. Degabriele et al. criticized [DFP15] the perfect decryptability condition required from the subverted ciphertext in BPR's model. Bellare et al. improved over the attacks in [BPR14], proposing stateless attacks [BJK15] against all randomized schemes. While previous attacks [BPR14, BJK15] targeted the encryption algorithm, Armour and Poettering proposed an attack [AP19b] by subverting the decryption algorithm. Hodges and Stebila explored the detectability of ASAs via state resetting [HS21]. Apart from these attacks on (authenticated) encryption schemes, ASAs have also been proposed on message authentication code [AP19a], signature schemes [AMV15, BSKC19, LCWW18], and key encapsulation mechanisms [CHY20]. Additionally, Russell et al. consider ASAs on (trapdoor) one-way functions and key generation, as well as a generic way to defend randomized algorithms against ASAs [RTYZ16, RTYZ17, RTYZ18].

ASAs were conceptualized to model government sponsored eavesdropping on real world protocols with millions of active users. So, it is just natural to explore these attacks against secure messaging services like WhatsApp and Telegram. Recently, Berndt et al. studied [BWP+20] the feasibility of ASAs on three popular protocols: TLS, WireGuard, and most notably Signal — the cryptographic protocol used in several messaging apps, including WhatsApp and Signal. To the best of our knowledge, such studies have not been conducted on Telegram's MTProto protocol.

1.3 Our Motivation

As pointed out by the Telegram security team in a private conversation, the code of all their official apps is open source and their builds are reproducible. This obviously makes massive subversion attacks against the Telegram official clients difficult to roll out. However, targeted attacks at individuals could still be deployed. Moreover, closed-source (or open source without reproducible builds) third-party clients would be easy to subvert. This second scenario is our main motivation in this work: is it possible to mount an efficient subversion attack against the authenticated encryption scheme that is used in Telegram?

1.4 Our Contributions

Our contributions are twofold.

First, we propose the first partial key recovery ASAs (see section 5) on the secret chat mode of Telegram. Our attacks are completely passive in nature and incur significantly less latency as compared to previous such attacks on generic authenticated encryption schemes [BJK15, AP19b]. Our attack exploits the random length padding used in the MTProto2.0 encryption. Strangely, each official client (desktop, android, iOS, tdlib library) uses different padding algorithms. Our attack can be mounted with the padding algorithm of the desktop client and the tdlib library (which can be used by third-party clients). As per our undetectability proofs (see Theorem 5.1 and 5.2), our subverted algorithms are indistinguishable from Telegram's original encryption algorithm from the desktop client or the tdlib library (depending on which one has been chosen for the corrupted client).

Second, we propose a minor change in the definition of MTProto2.0, that ensures all the advantages of the existing algorithm, and thwarts the proposed key recovery attack. In fact, we show that the modified algorithm is subversion-resistant in most of the practical scenarios. This is done in three steps. First, we show that an abstraction of MTProto2.0, called MTProto-G, is a secure deterministic authenticated encryption (DAE) scheme. Second, we make three small changes, mainly in the padding algorithm of MTProto2.0, to make the protocol deterministic. Finally, under the assumptions of perfect decryptability and key-independent messages, we show that the modified protocol, called MTProto-D, is subversion-resistant in context of algorithm substitution attacks, resulting in a more secure solution for Telegram.

Responsible Disclosure to Telegram: We followed the standard responsible disclosure policy and reported our findings to the Telegram security team in August 2021, along with our suggestion to drop the randomness from the padding algorithm. As noted above, they countered our findings by noting that the official Telegram apps are open source and support reproducible builds that can be verified by independent researchers who regularly audit the security of Telegram apps. In addition, they also asserted that cryptographic keys can be leaked through various side channels as well. We pointed out that targeted attacks at individuals are still a concerning possibility, and that the presence of side channels only accentuates the impact of our attack, but the Telegram security team did not believe it was a meaningful threat. We concluded our exchange with the Telegram team by mentioning the issue of closed-source (or open source without reproducible builds) third-party Telegram-compatible clients which will still be vulnerable to such mass-surveillance, unless the algorithm is updated. Subsequently, in early December 2023, we also informed several popular third-party Telegram-compatible clients about our findings.

2 Preliminaries

Notational Setup: For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$, and $\{0,1\}^n$ denotes the set of bit strings of length n. The set of all bit strings (including the empty string) is denoted $\{0,1\}^*$, and $|X|$ denotes the number of bits in $X \in \{0,1\}^*$. For any integer m, $\{0,1\}^{\leq m}$ denotes the set of all bit strings of bit length at most m. For any two bit strings M and M', we denote by $M||M'$ the concatenation of M and M'. For any bit string z we write $z[a,b]$ for the sub-string of z from bit a to bit b (inclusive). For $i, m \in \mathbb{N}$ such that $i < 2^m$, we define $\langle i \rangle_m$ as the m-bit little endian encoding of the integer i.

The set of all functions from $\mathcal{X}$ to $\mathcal{Y}$ is denoted $\mathcal{F}(\mathcal{X}, \mathcal{Y})$, and the set of all permutations of $\mathcal{X}$ is denoted $\mathcal{P}(\mathcal{X})$. Extending notation, for a finite set $\mathcal{T}$, we denote by $\widetilde{\mathsf{Perm}}(\mathcal{T}, \mathcal{X})$ the set of all *indexed permutations*, families of permutations $\pi_t \in \mathcal{P}(\mathcal{X})$, indexed by $t \in \mathcal{T}$. For a finite set $\mathcal{X}$, $\mathsf{X} \leftarrow_{\$} \mathcal{X}$ denotes the uniform at random sampling of X from $\mathcal{X}$.

Adversary: A (q,t)-adversary D is an interactive algorithm with access to an oracle, that makes at most q oracle queries, runs in time at most t, and returns an output at the end. In addition, we note that all the security notions in this paper are given in a general multi-user security setting where the adversary can query any one of several independent instances of the oracle at hand.

(Tweakable) Block Cipher: A *block cipher* family E, with key space $\mathcal{K}$ and block space $\{0,1\}^n$, is a function family $E = \{F_k : \{0,1\}^n \to \{0,1\}^n\}_{k \in \mathcal{K}}$, such that for any key $k \in \mathcal{K}$, $E_k(\cdot)$ is a permutation of $\{0,1\}^n$. We write $E_k^-(\cdot)$ to denote the inverse of $E_k(\cdot)$.

A tweakable block cipher family $\widetilde{E}$, with key space $\mathcal{K}$, tweak space $\mathcal{T}$, and block space $\{0,1\}^n$, is a function family $\widetilde{E} = \{F_k : \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n\}_{k \in \mathcal{K}}$, such that for any key $k \in \mathcal{K}$ and tweak $t \in \mathcal{T}$, $\widetilde{E}_k(t, \cdot)$ is a permutation of $\{0,1\}^n$. We write $\widetilde{E}_k^-(t, \cdot)$ to denote the inverse of $\widetilde{E}_k(t, \cdot)$.

Strong Pseudorandom Permutation: The security of any block cipher is formalized via the notion of a *strong pseudorandom permutation* or SPRP game. We have illustrated the SPRP game in the general multi-user setting in Figure 2.1. The advantage of any adversary D against a block cipher E is defined as

$$\mathbf{Adv}_E^{\mathsf{sprp}}(D) = \left| \Pr\left(\mathrm{Re}_E^{\mathrm{sprp}}(D) = 1\right) - \Pr\left(\mathrm{Id}_E^{\mathrm{sprp}}(D) = 1\right) \right|.$$

We sometimes also use the restricted notion of Pseudorandom permutation (PRP), where the adversary is only given access to O^+. Formally, we define the PRP advantage of D

$\underline{\mathrm{Re}_E^{\mathrm{sprp}}(D)\ \boxed{\mathrm{Id}_E^{\mathrm{sprp}}(D)}}$
$u \leftarrow 0$
$b \leftarrow D^{\textsc{New},\mathsf{O}^{\pm}}$
return $b =_? 1$

Oracle $\textsc{New}()$
$u \leftarrow u + 1$
$K_u \leftarrow_\$ \mathcal{K}$
$\boxed{\Pi_u \leftarrow_\$ \mathcal{P}(\{0,1\}^n)}$

Oracle $\mathsf{O}^+(x, i)$
if $i \notin [u]$ **then**
  return $\perp$
$y \leftarrow E_{K_i}(x)$
$\boxed{y \leftarrow \Pi_i(x)}$
return y

Oracle $\mathsf{O}^-(y, i)$
if $i \notin [u]$ **then**
  return $\perp$
$x \leftarrow E^-_{K_i}(y)$
$\boxed{x \leftarrow \Pi^-_i(y)}$
return x

Figure 2.1: Strong pseudorandom permutation game.

$\underline{\mathrm{Re}_{\widetilde{E}}^{\mathrm{tprp}}(D)\ \boxed{\mathrm{Id}_{\widetilde{E}}^{\mathrm{tprp}}(D)}}$
$u \leftarrow 0$
$b \leftarrow D^{\textsc{New},\mathsf{O}}$
return $b =_? 1$

Oracle $\textsc{New}()$
$u \leftarrow u + 1$
$K_u \leftarrow_\$ \mathcal{K}$
$\boxed{\widetilde{\Pi}_u \leftarrow_\$ \widetilde{\mathsf{Perm}}(\mathcal{T}, \{0,1\}^n)}$

Oracle $\mathsf{O}^+(t, x, i)$
if $i \notin [u]$ **then**
  return $\perp$
$y \leftarrow \widetilde{E}_{K_i}(t, x)$
$\boxed{y \leftarrow \widetilde{\Pi}_i(t, x)}$
return y

Figure 2.2: Tweakable pseudorandom permutation game.

against E as

$$\mathbf{Adv}_E^{\mathsf{prp}}(D) = \left|\Pr\left(\mathrm{Re}_E^{\mathrm{prp}}(D) = 1\right) - \Pr\left(\mathrm{Id}_E^{\mathrm{prp}}(D) = 1\right)\right|,$$

where $\mathrm{Re}_E^{\mathrm{prp}}(D)$ and $\mathrm{Id}_E^{\mathrm{prp}}(D)$ are defined similarly to their SPRP counterparts, except for the restriction that D cannot query O^-.

Tweakable Pseudorandom Permutation: The security of any tweakable block cipher is formalized via the notion of a *tweakable pseudorandom permutation* or TPRP game. We have illustrated the TPRP game in the general multi-user setting in Figure 2.2. The advantage of any adversary D against a tweakable block cipher $\widetilde{E}$ is defined as

$$\mathbf{Adv}_{\widetilde{E}}^{\mathsf{tprp}}(D) = \left|\Pr\left(\mathrm{Re}_{\widetilde{E}}^{\mathrm{tprp}}(D) = 1\right) - \Pr\left(\mathrm{Id}_{\widetilde{E}}^{\mathrm{tprp}}(D) = 1\right)\right|.$$

Pseudorandom Function: The security of any keyed function is formalized via the notion of a *pseudorandom function* or PRF game, illustrated for the general multi-user setting in Figure 2.3. The PRF advantage of any adversary D against a keyed function family $F = \{F_{\mathsf{K}} : \mathcal{X} \to \mathcal{Y}\}_{\mathsf{K} \in \mathcal{K}}$ is defined as

$$\mathbf{Adv}_F^{\mathsf{prf}}(D) = \left|\Pr\left(\mathrm{Re}_F^{\mathrm{prf}}(D) = 1\right) - \Pr\left(\mathrm{Id}_F^{\mathrm{prf}}(D) = 1\right)\right|.$$

A weaker version of the PRF notion does not allow arbitrary queries from the distinguisher. Instead, the distinguisher receives a uniform and independent random input, and the

$\underline{\mathrm{Re}_F^{\mathrm{prf}}(D)\ \boxed{\mathrm{Id}_F^{\mathrm{prf}}(D)}}$
$u \leftarrow 0$
$b \leftarrow D^{\textsc{New},\mathsf{O}}$
return $b =_? 1$

Oracle $\textsc{New}()$
$u \leftarrow u + 1$
$K_u \leftarrow_\$ \mathcal{K}$
$\boxed{\Gamma_u \leftarrow_\$ \mathcal{F}(\mathcal{X}, \mathcal{Y})}$

Oracle $\mathsf{O}(x, i)$
if $i \notin [u]$ **then**
  return $\perp$
$y \leftarrow F_{K_i}(x)$
$\boxed{y \leftarrow \Gamma_i(x)}$
return y

Figure 2.3: Pseudorandom function game.

$\underline{\mathrm{Re}_{\mathcal{E}}^{\mathrm{priv\$}}(D)}$ $\boxed{\underline{\mathrm{Id}_{\mathcal{E}}^{\mathrm{priv\$}}(D)}}$

$u \leftarrow 0$
$b \leftarrow D^{\textsc{New},\mathsf{O}}$
return $b =_? 1$

Oracle $\textsc{New}()$

$u \leftarrow u + 1$
$K_u \leftarrow_\$ \mathcal{K}$
$\boxed{\Gamma_u \leftarrow_\$ \mathcal{F}_{\mathsf{lp}}(\mathcal{R} \times \mathcal{M}, \mathcal{M})}$

Oracle $\mathsf{O}(m, i)$

if $i \notin [u]$ **then**
  return $\perp$
$R \leftarrow_\$ \mathcal{R}$
$c \leftarrow \mathsf{E}^+_{K_i,R}(m)$
$\boxed{c \leftarrow \Gamma_i(R, m)}$
return (R, c)

Figure 2.4: Priv\$ game. Here $\mathcal{F}_{\mathsf{lp}}(\mathcal{R} \times \mathcal{M}, \mathcal{M})$ denotes the set of all $f : \mathcal{R} \times \mathcal{M}$ to $\mathcal{M}$ such that $|f(r, m)| = |m|$ for all $(r, m) \in \mathcal{R} \times \mathcal{M}$.

evaluation of the underlying function over this input, whenever it pings the oracle. The weak PRF notion can also be interpreted as a game where adversary is constrained to sample its input uniformly at random. Formally, we define the wPRF advantage of D against F as

$$\mathbf{Adv}_F^{\mathsf{wprf}}(D) = \left|\Pr\left(\mathrm{Re}_F^{\mathrm{prf}}(D^\$) = 1\right) - \Pr\left(\mathrm{Id}_F^{\mathrm{prf}}(D^\$) = 1\right)\right|,$$

where $D^\$$ is used to denote the fact that D samples its queries uniformly at random at each turn.

IV-based Encryption: A $(\mathcal{K}, \mathcal{R}, \mathcal{M})$-encryption scheme E is a tuple of algorithms $(\mathsf{E}^+, \mathsf{E}^-)$, defined over the key space $\mathcal{K}$, IV space $\mathcal{R}$, message and ciphertext space $\mathcal{M}$, where

$$\mathsf{E}^+ : \mathcal{K} \times \mathcal{R} \times \mathcal{M} \to \mathcal{M} \qquad \mathsf{E}^- : \mathcal{K} \times \mathcal{R} \times \mathcal{M} \to \mathcal{M}.$$

For all $(k, r) \in \mathcal{K} \times \mathcal{R}$, $\mathsf{E}^-_{k,r}(\cdot) \coloneqq \mathsf{E}^-(k, r, \cdot)$, referred as the decryption algorithm, is defined as the inverse of $\mathsf{E}^+_{k,r}(\cdot) \coloneqq \mathsf{E}^+(k, r, \cdot)$, referred as the encryption algorithm, i.e., for all $m \in \mathcal{M}$, $\mathsf{E}^-_{k,r}(\mathsf{E}^+_{k,r}(m)) = m$. It is not necessary to release the IV along with the ciphertext if the IV can be derived from the sequence number or the traffic secret (see e.g., TLS 1.3). Without loss of generality we assume that the IV is released along with the ciphertext in order to facilitate correct decryption. In most cases, including this work, $\mathsf{E}^+_{k,r}(\cdot)$ is a length-preserving permutation for all $(k, r) \in \mathcal{K} \times \mathcal{R}$. In this work we only consider random IV schemes, i.e., the IV is sampled uniformly at random for each execution of the encryption algorithm.

The security of any IV-based encryption scheme is formalized via the notion of a Privacy game, illustrated for the general multi-user setting in Figure 2.4. The privacy advantage of D against E is defined as

$$\mathbf{Adv}_{\mathsf{E}}^{\mathsf{priv\$}}(D) \coloneqq \left|\Pr\left(\mathrm{Re}_{\mathsf{E}}^{\mathrm{priv\$}}(D) = 1\right) - \Pr\left(\mathrm{Id}_{\mathsf{E}}^{\mathrm{priv\$}}(D) = 1\right)\right|,$$

Deterministic Authenticated Encryption: A $(\mathcal{K}, \mathcal{A}, \mathcal{M}, \mathcal{T})$-deterministic authenticated encryption scheme $\mathcal{E}$ is a tuple of algorithms $(\mathcal{E}^+, \mathcal{E}^-)$ defined over the key space $\mathcal{K}$, associated data space $\mathcal{A}$, message and ciphertext space $\mathcal{M}$, and tag space $\mathcal{T}$, where:

$$\mathcal{E}^+ : \mathcal{K} \times \mathcal{A} \times \mathcal{M} \to \mathcal{T} \times \mathcal{M} \quad \mathcal{E}^- : \mathcal{K} \times \mathcal{A} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M} \cup \{\perp\},$$

and $\perp$ denotes the error symbol indicating authentication failure. For all keys $k \in \mathcal{K}$, we write $\mathcal{E}^+_k(\cdot, \cdot) \coloneqq \mathcal{E}^+(k, \cdot, \cdot)$, referred as the encryption algorithm, and $\mathcal{E}^-_k(\cdot, \cdot) \coloneqq \mathcal{E}^-(k, \cdot, \cdot)$, referred as the decryption algorithm. For correct decryption, it is required that $\mathcal{E}^-_k(a, \mathcal{E}^+_k(a, m)) = m$ for all $(k, a, m) \in \mathcal{K} \times \mathcal{A} \times \mathcal{M}$.

$\underline{\mathrm{Re}^{\mathsf{dae}}_{\mathcal{E}}(D)}$ $\boxed{\underline{\mathrm{Id}^{\mathsf{dae}}_{\mathcal{E}}(D)}}$

$u \leftarrow 0$

$b \leftarrow D^{\textsc{New},\mathsf{O}^{\pm}}$

return $b =_? 1$

Oracle $\textsc{New}()$

$u \leftarrow u + 1$

$K_u \leftarrow\!\!\$\ \mathcal{K}$

$\boxed{\Gamma_u \leftarrow\!\!\$\ \mathcal{F}_{\mathcal{E}}}$

Oracle $\mathsf{O}^+(a, m, i)$

if $i \notin [u]$ **then**

  return $\perp$

$(t, c) \leftarrow \mathcal{E}^+_{K_i}(a, m)$

$\boxed{(t, c) \leftarrow \Gamma_i(a, m)}$

return (t, c)

Oracle $\mathsf{O}^-(a, t, c, i)$

if $i \notin [u]$ **then**

  return $\perp$

$m \leftarrow \mathcal{E}^-_{K_i}(a, t, c)$

$\boxed{m \leftarrow \perp}$

return m

Figure 2.5: Deterministic authenticated encryption game, where $\mathcal{F}_{\mathcal{E}} = \mathcal{F}(\mathcal{A} \times \mathcal{M}, \mathcal{T} \times \mathcal{M})$. In order to avoid trivial wins, the adversary is not allowed to query oracle O^- with an answer he received from a query to O^+.

The security of any deterministic authenticated encryption scheme is formalized via the notion of a DAE game, illustrated for the general multi-user setting in Figure 2.5. The DAE advantage of D against $\mathcal{E}$ is defined as

$$\mathbf{Adv}^{\mathsf{dae}}_{\mathcal{E}}(D) \coloneqq \left|\Pr\left(\mathrm{Re}^{\mathrm{dae}}_{\mathcal{E}}(D) = 1\right) - \Pr\left(\mathrm{Id}^{\mathrm{dae}}_{\mathcal{E}}(D) = 1\right)\right|.$$

2.1 Subversion Attacks

We formalize subversion attacks by following the definitions from [BJK15]. From a high level, a subversion attack aims to replace an encryption scheme with a different keyed algorithm, with the following two goals:

- the subversion should be difficult to distinguish from the actual encryption scheme for someone who does not know the adversary's key;
- the subversion should break the security of the subverted encryption scheme in some way.

In this work, as in [BJK15], we focus on key-recovery attacks.

Let $\mathcal{E} = (\mathcal{E}^+, \mathcal{E}^-)$ be a symmetric (authenticated) encryption scheme with key space $\mathcal{K}$. A subversion of $\mathcal{E}$ is a tuple $\widetilde{\mathcal{E}} = (\widetilde{\mathcal{K}}, \widetilde{\mathcal{E}^+}, \widetilde{\mathcal{E}^{\mathrm{ext}}})$, where the master-key space $\widetilde{\mathcal{K}}$ is a non-empty set, such that:

- the subverted encryption algorithm $\widetilde{\mathcal{E}^+}$ maps a tuple (K_A, K_E, A, M, σ) to a pair (C, σ'), where $A, M \in \{0,1\}^*$, $K_A \in \widetilde{\mathcal{K}}$, $K_E \in \mathcal{K}$, C is a ciphertext and σ' corresponds to the update of the state σ;
- the key-recovery algorithm $\widetilde{\mathcal{E}^{\mathrm{ext}}}$ takes as input a master key $\widetilde{K}$, a vector of associated data $\mathbf{A}$, a vector of ciphertexts $\mathbf{C}$, and produces a key guess $K \in \mathcal{K}$.

We say that $\widetilde{\mathcal{E}}$ is *decryptable* (with respect to $\mathcal{E}$) if, for every plaintext M, every associated data A, every key tuple $(K_E, K_A) \in \mathcal{K} \times \widetilde{\mathcal{K}}$ and every state σ, one has

$$\mathcal{E}^-_{K_E}(A, \widetilde{\mathcal{E}^+}_{K_A, K_E}(A, M, \sigma)) = M.$$

Besides, if the state σ is never updated by the encryption algorithm, we say that $\widetilde{\mathcal{E}}$ is *stateless*. Otherwise, it is said to be *stateful*.

Undetectability: Clearly, a subversion attack can only be effective as long as it is hard to detect. In this section, we focus on computational detection: the output of the attacker's encryption scheme should be indistinguishable from the output of the subverted scheme, even from the point of view of the decryption algorithm. We formalize this notion with the (multi-user) detection games presented in Figure 2.6. Our Undetectability definition slightly differs from the Strong Undetectability notion from [BJK15] in two ways:

$\mathrm{Re}^{\mathrm{det}}_{\mathcal{E},\widetilde{\mathcal{E}}}(D)$ $\boxed{\mathrm{Sub}^{\mathrm{det}}_{\mathcal{E},\widetilde{\mathcal{E}}}(D)}$

$u \leftarrow 0$
$K_A \leftarrow\!\!\$\ \widetilde{\mathcal{K}}$
$b \leftarrow D^{\textsc{New},\textsc{Enc}}()$
return $b =_? 1$

Oracle $\textsc{New}()$

$u \leftarrow u + 1$
$\sigma_u \leftarrow \epsilon$
$K_u \leftarrow\!\!\$\ \mathcal{K}$
return (K_u, σ_u)

Oracle $\textsc{Enc}(A, M, i)$

if $i \notin [u]$ **then**
  return $\perp$
$(C, \sigma_i) \leftarrow \mathcal{E}^+_{K_i}(A, M, \sigma_i)$
$\boxed{(C, \sigma_i) \leftarrow \widetilde{\mathcal{E}^+}_{K_A,K_i}(A, M, \sigma_i)}$
return (C, σ_i)

Figure 2.6: Games used to define detection of subversion $\widetilde{\mathcal{E}}$ of encryption scheme $\mathcal{E}$.

- we allow the subverted algorithm to also be stateful: looking ahead momentarily, our goal is to model the behavior of secret chats in the MTProto protocol, which maintain a state σ that counts the number of sent and received messages for each key (see Section 4.2);
- in [BJK15], the attacker is allowed to choose the key of the encryption scheme $\mathcal{E}$, while, in our game, the key is generated uniformly at random.

In a sense, we can think of our model as a Strong Undetectability notion in the Honest setting, where the adversary is assumed to honestly generate encryption keys. We argue that this restriction is natural. In our scenario, the attacker will either be Telegram servers, a Telegram client, or an external observer that is trying to detect an ASA. These actors all have an interest in keeping communications secure by generating the encryption keys uniformly at random, instead of intentionally generating weak keys (for example by using small order elements in a Diffie-Hellman key exchange in order to create a lot of key collisions).

As usual, we measure the advantage of an algorithm D trying to distinguish between the genuine encryption scheme $\mathcal{E}$ and its subversion $\widetilde{\mathcal{E}}$ as follows:

$$\mathbf{Adv}^{\mathsf{det}}_{\mathcal{E},\widetilde{\mathcal{E}}}(D) \coloneqq \left|\Pr\left(\mathrm{Re}^{\mathrm{det}}_{\mathcal{E},\widetilde{\mathcal{E}}}(D) = 1\right) - \Pr\left(\mathrm{Sub}^{\mathrm{det}}_{\mathcal{E},\widetilde{\mathcal{E}}}(D) = 1\right)\right|.$$

Key Recovery: The goal of a subversion attack is to break the security of the original encryption scheme $\mathcal{E}$ in some way. The weakest possible goal would be to allow the attacker to distinguish between $\widetilde{\mathcal{E}}$ and an ideal encryption scheme. However, the practical consequences of such an attack are small. Instead, we focus on attacks that allow the recovery of part of the key of $\mathcal{E}$, in order to allow the decryption of all ciphertexts.

Following [BJK15], we formalize the Key Recovery experiment in Figure 2.7. Note that the game is parameterized by an algorithm $\mathcal{M}$ that samples new message queries when given the current state σ' (which may be different from the state maintained by the protocol), and a number of queries q. We stress that our attack will work independently of the choice of $\mathcal{M}$, and its success will only depend on the number of encryption queries. The subversion attack is successful if $\widetilde{\mathcal{E}^{\mathrm{ext}}}$ recovers the key K_E from the ciphertexts produced by $\widetilde{\mathcal{E}^+}$ on messages produced by $\mathcal{M}$, and its advantage is defined as:

$$\mathbf{Adv}^{\mathsf{kr}}_{\mathcal{M},q}(\widetilde{\mathcal{E}}) \coloneqq \Pr\left(\mathrm{Kr}_{\widetilde{\mathcal{E}}}(\mathcal{M}, q) = 1\right)$$

Previous Subversion Attacks: In [BPR14], Bellare et al. present a very simple subversion attack again IV-based encryption schemes such that the IV is public in ciphertexts (see Algorithm 2.1). It subversion attack simply encrypts the target key K_E using the adversarial key K_A, and uses this value as IV for the first encryption query. The main drawback of this attack is that it is inherently stateful: a simple state reset allows the detection of the subversion, as it triggers an IV repetition. In order to avoid such simple countermeasures and to make the attack usable against any randomized encryption scheme,

$\underline{\mathrm{Kr}_{\widetilde{\mathcal{E}}}(\mathcal{M}, q)}$

$(K_E, K_A) \leftarrow\!\!\$\ \mathcal{K} \times \widetilde{\mathcal{K}}$
$K \leftarrow \widetilde{\mathcal{E}^{\mathrm{ext}}}(K_A, \mathcal{A}, \textsc{Enc}(\mathcal{M}, q))$
return $K =_?\ K_E$

$\underline{\text{Oracle } \textsc{Enc}(\mathcal{M}, q)}$

$\sigma' \leftarrow \varepsilon$
for $i \in \{1, \ldots, q\}$
  $(A_i, M_i, \sigma') \leftarrow \mathcal{M}(i, \sigma')$
  $(C_i, \sigma') \leftarrow \widetilde{\mathcal{E}^{+}}_{K_A, K_E}(A_i, M_i, \sigma')$
return (A_i, C_i)

Figure 2.7: Games used to define the key recovery experiment of the subversion $\widetilde{\mathcal{E}}$ of encryption scheme $\mathcal{E}$.

Bellare et al. introduce a new stateless substitution attack in [BJK15] (see Algorithm 2.2 for a description of the algorithm). The key idea is to rely on a second PRF F with output space $\{0,1\} \times \{1, \ldots, n\}$ where n denotes the length of K_E, and to sample IVs until the corresponding ciphertext C satisfies $F_{K_A}(C) = (b, i)$, where $K_E[i] = b$. No state is needed anymore, but this comes at a cost: the attack is now randomized, and can fail to transmit particular key bit. More general subversion attacks have been introduced later. As an example, Armour and Poettering [AP19b] proposed another stateless attack that targets the decryption algorithm of any authenticated encryption scheme. It works similarly to the attack from [BJK15]: when the decryption algorithm is given a ciphertext such that $F_{K_A}(C) = (b, i)$, where $K_E[i] = b$, it will reject the ciphertext instead of decrypting. The main difference with previous attacks is that this subversion comes at a functionality cost: some valid ciphertexts get rejected. In order to avoid the easy detection of the attack, the subverted algorithm will only test a small fraction of all ciphertexts.[1]

Algorithm 2.1 Pseudocode of the subversion attack from [BPR14]. Here $\mathcal{E}$ denotes an IV-based encryption scheme, and E is a length-preserving deterministic encryption scheme.

function $\widetilde{\mathcal{E}^{+}}(K_A, K, A, M, \sigma)$
  if $\sigma = 0$ **then**
    $\mathrm{iv} \leftarrow E(K_A, K)$
  else
    $\mathrm{iv} \leftarrow\!\!\$\ \{0,1\}^n$
  end if
  $C \leftarrow \mathcal{E}^{+}(K, A, M, \mathrm{iv})$
  $\sigma \leftarrow \sigma + 1$
  return (C, σ)
end function

function $\widetilde{\mathcal{E}^{\mathrm{EXT}}}((K_A, \mathbf{A}, \mathbf{C}, i))$
  $\mathrm{iv} \leftarrow \mathrm{iv}(\mathbf{C}[1])$
  $K \leftarrow E^{-}(K_A, \mathrm{iv})$
  return K
end function

3 MTProto

Telegram clients rely on the MTProto protocol to secure communications. A message that is typed by a user, or any application-defined message, first has to be *enriched* to include additional information, along with a padding and some random bits; we refer to these as protocol-enriched messages. Then, these plaintexts are encrypted using a DAE scheme that is also dubbed MTProto. In this section, we focus on the description of the DAE algorithm, while Section 4 is devoted to the description of the whole protocol.

[1] This is done at random by sampling a Bernoulli random variable in order to decide whether to attack a particular ciphertext or not.

Algorithm 2.2 Pseudocode of the subversion attack from [BJK15]. Here $\mathcal{E}$ denotes encryption scheme that uses n-bit IVs, E is a length-preserving deterministic encryption scheme, and F a PRF with range $\{0,1\} \times \{1,\ldots,k\}$, where k denotes the key size of $\mathcal{E}$. The state σ is constant ($\sigma = \epsilon$), meaning that the attack is stateless.

function $\widetilde{\mathcal{E}^{+}}_s(K_A, K_E, M, A, \sigma)$
  $j \leftarrow 0$
  do
    $j \leftarrow j + 1$
    $r \leftarrow_{\$} \{0,1\}^n$
    $C \leftarrow \mathcal{E}^{+}(K_E, A, M, r)$
    $(v, t) \leftarrow F(K_A, C)$
  while $(j < s)$ and $K_E[t] \neq v$
  return (C, ϵ)
end function

function $\widetilde{\mathcal{E}^{\text{EXT}}}((K_A, \mathbf{C}, \mathbf{A}, i))$
  $K \leftarrow 0^k$
  for $i = 1, \ldots, |\mathbf{C}|$ **do**
    $(v, t) \leftarrow F(K_A, C)$
    $K[t] = v$
  end for
  return K
end function

3.1 Generic View of MTProto

MTProto can be viewed as a somewhat generic deterministic authenticated encryption scheme, referred here as MTProto-G, that utilizes two independently keyed functions, $\mathsf{F} : \mathcal{K}_1 \times \{0,1\}^* \to \{0,1\}^\tau$ and $\mathsf{G} : \mathcal{K}_2 \times \{0,1\}^\tau \to \{0,1\}^{\kappa+n}$, and a $(\{0,1\}^\kappa, \{0,1\}^n, \mathcal{M})$-encryption scheme E. As illustrated in Figure 3.1, the output of F serves two purposes:

1. obviously it acts as the authentication tag;
2. it acts as the input for deriving the keys and initialization values for the encryption scheme E.

Lemma 3.1. *Let* MTProto-G *be defined as above. For* $\mu, q_{\max}, q, \ell, \sigma, t > 0$*, let* $\mathscr{A}$ *be a* $(\mu, q_{\max}, q, \ell, \sigma, t)$*-distinguisher against* MTProto-G *that runs in time at most* t*, and issues at most* q *queries, of length at most* ℓ n*-bit blocks, for a total queries length* μ*, over at most* μ *users, and such that each user is queried at most* $q_{\max}$*. Then, there exist* $(\mu, q_{\max}, q, \ell, \sigma, t')$*,* $(\mu, q_{\max}, q, \hat{t})$*, and* $(q, 1, q, \ell, \sigma, \dot{t})$ *distinguishers* $\mathscr{B}$*,* $\mathscr{C}$*, and* $\mathscr{D}$*, respectively, such that*

$$\mathbf{Adv}^{\mathsf{dae}}_{\mathsf{MTProto\text{-}G}}(\mathscr{A}) \leq \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{F}}(\mathscr{B}) + \mathbf{Adv}^{\mathsf{wprf}}_{\mathsf{G}}(\mathscr{C}) + \mathbf{Adv}^{\mathsf{priv\$}}_{\mathsf{E}}(\mathscr{D}) + \frac{q}{2^\tau} + \frac{q^2}{2^\tau}, \tag{1}$$

where $t' = O(t + qt_{\mathsf{F}})$*,* $\hat{t} = O(t + qt_{\mathsf{G}})$*, and* $\dot{t} = O(t + qt_{\mathsf{E}})$*.*

Proof. First of all, we view MTProto-G as an instance of the SIV paradigm [RS06], where F is used to generate the tag (also acts as the synthetic IV), and the combination of G and E is viewed as an IV-based encryption scheme. More formally, we define a $(\mathcal{K}, \{0,1\}^\tau, \mathcal{M})$-encryption scheme $\overline{\mathsf{E}}$ (also see Figure 3.1) as follows: for all $k, t, m \in \mathcal{K} \times \{0,1\}^\tau \times \mathcal{M}$, we have

$$(l, \text{iv}) := \mathsf{G}_k(t) \qquad c := \mathsf{E}^{+}_{l,\text{iv}}(m) \qquad \overline{\mathsf{E}}[\mathsf{G}, \mathsf{E}]^{+}_{k,t}(m) := c$$

Then, using the SIV composition result by Rogaway and Shrimpton [RS06, Theorem 2], we have

$$\mathbf{Adv}^{\mathsf{dae}}_{\mathsf{MTProto\text{-}G}}(\mathscr{A}) \leq \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{F}}(\mathscr{B}) + \mathbf{Adv}^{\mathsf{priv\$}}_{\overline{\mathsf{E}}}(\mathscr{A}') + \frac{q}{2^\tau},$$

where $\mathscr{A}'$ is a $(\mu, q_{\max}, q, \ell, \sigma, t'')$-distinguisher for $t'' = O(t + qt_{\overline{\mathsf{E}}})$. Note that, the generic reduction result in [RS06] is proved in single-key setting. However, exactly the same

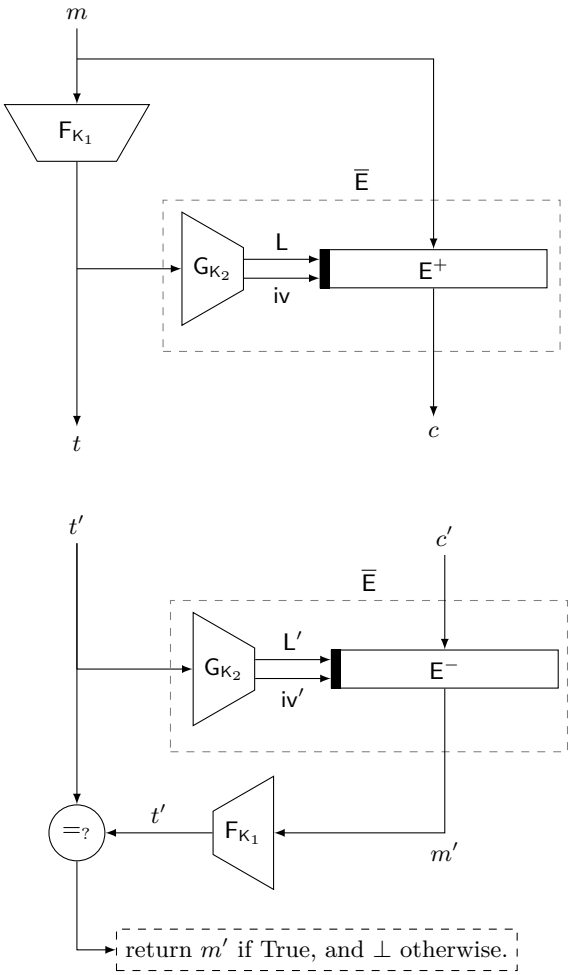

Figure 3.1: Encryption (**top**) and decryption (**bottom**) algorithms in MTProto-G. The dashed rectangle represents the IV-based encryption scheme $\overline{\mathsf{E}}$.

approach generalizes to the multi-user setting as well. Next, by definition, one has

$$\begin{aligned}\mathbf{Adv}^{\mathsf{priv\$}}_{\overline{\mathsf{E}}}(\mathscr{A}') &= \left|\Pr_{(\mathsf{K}_i)_{i\in[\mu]} \xleftarrow{\$} \mathcal{K}}\left(\mathscr{A}'^{(\overline{\mathsf{E}}[\mathsf{G}_{\mathsf{K}_i},\mathsf{E}]^+)_{i\in[\mu]}}=1\right) - \Pr_{\$}\left(\mathscr{A}'^{\$}=1\right)\right| \\ &\le \left|\Pr_{(\mathsf{K}_i)_{i\in[\mu]} \xleftarrow{\$} \mathcal{K}}\left(\mathscr{A}'^{(\overline{\mathsf{E}}[\mathsf{G}_{\mathsf{K}_i},\mathsf{E}]^+)_{i\in[\mu]}}=1\right)\right. \\ &\quad \left. - \Pr_{(\Gamma_i)_{i\in[\mu]} \xleftarrow{\$} \mathcal{F}(\tau,\kappa+n)}\left(\mathscr{A}'^{(\overline{\mathsf{E}}[\mathsf{\Gamma}_i,\mathsf{E}]^+)_{i\in[\mu]}}=1\right)\right| \\ &\quad + \left|\Pr_{(\Gamma_i)_{i\in[\mu]} \xleftarrow{\$} \mathcal{F}(\tau,\kappa+n)}\left(\mathscr{A}'^{(\overline{\mathsf{E}}[\mathsf{\Gamma}_i,\mathsf{E}]^+)_{i\in[\mu]}}=1\right) - \Pr_{\$}\left(\mathscr{A}'^{\$}=1\right)\right|. \quad (2)\end{aligned}$$

Now, all that remains is to show that there exists a $(\mu, q_{\max}, q, \hat{t})$-distinguisher $\mathscr{C}$ and a $(q, 1, q, \ell, \sigma, \dot{t})$-distinguisher $\mathscr{D}$ such that the first absolute difference on the right hand side is bounded by $\mathbf{Adv}^{\mathsf{wprf}}_{\mathsf{G}}(\mathscr{C})$ and the second difference is bounded by $\mathbf{Adv}^{\mathsf{priv\$}}_{\mathsf{E}}(\mathscr{D}) + q^2/2^\tau$.

First, we construct the distinguisher $\mathscr{C}$, which is trying to distinguish between $(\mathsf{G}_{\mathsf{K}_i}(\$))_{i\in[\mu]}$ and $(\mathsf{\Gamma}_i(\$))_{i\in[\mu]}$, where \$ denotes the uniform distribution sampler implemented via a uniform random function from $\mathcal{M}$ to $\mathcal{T}$ (courtesy $\mathbf{Adv}^{\mathsf{prf}}_{\mathsf{F}}(\mathscr{B})$). We simply define $\mathscr{C}$ as the distinguisher that runs $\mathscr{A}'$ in a black box manner, answering all its queries by applying E^+ (keyed with the answers given by its own oracle on uniform at random inputs) and outputs the same value as $\mathscr{A}'$. Then, clearly, $\mathscr{C}$ correctly simulates $(\overline{\mathsf{E}}[\mathsf{G}_{\mathsf{K}_i}(\$),\mathsf{E}]^+)_{i\in[\mu]}$ when its oracle is $(\mathsf{G}_{\mathsf{K}_i}(\$))_{i\in[\mu]}$, and it correctly simulates $(\overline{\mathsf{E}}[\mathsf{\Gamma}_i,\mathsf{E}]^+)_{i\in[\mu]}$ when its oracle is $(\mathsf{\Gamma}_i(\$))_{i\in[\mu]}$. Moreover, $\mathscr{C}$ makes at most q queries to its oracle and runs in time $\hat{t} = O(t + qt_{\mathsf{G}})$. Thus, we have

$$\begin{aligned}\mathbf{Adv}^{\mathsf{wprf}}_{\mathsf{G}}(\mathscr{C}) &= \left|\Pr_{\$,\substack{(\mathsf{K}_i)_{i\in[\mu]}\\ \xleftarrow{\$}\mathcal{K}}}\left(\mathscr{C}^{(\mathsf{G}_{\mathsf{K}_i}(\$))_{i\in[\mu]}}=1\right) - \Pr_{\$,\substack{(\mathsf{\Gamma}_i)_{i\in[\mu]}\xleftarrow{\$}\\ \mathcal{F}(\tau,\kappa+n)}}\left(\mathscr{C}^{(\mathsf{\Gamma}_i(\$))_{i\in[\mu]}}=1\right)\right| \\ &\ge \left|\Pr_{\substack{(\mathsf{K}_i)_{i\in[\mu]}\\ \xleftarrow{\$}\mathcal{K}}}\left(\mathscr{A}'^{(\overline{\mathsf{E}}[\mathsf{G}_{\mathsf{K}_i},\mathsf{E}]^+)_{i\in[\mu]}}=1\right) - \Pr_{\substack{(\mathsf{\Gamma}_i)_{i\in[\mu]}\xleftarrow{\$}\\ \mathcal{F}(\tau,\kappa+n)}}\left(\mathscr{A}'^{(\overline{\mathsf{E}}[\mathsf{\Gamma}_i,\mathsf{E}]^+)_{i\in[\mu]}}=1\right)\right|. \quad (3)\end{aligned}$$

Before we move on to constructing the distinguisher $\mathscr{D}$, we introduce a small change in the game: instead of sampling the IVs for $\mathscr{A}'$'s oracle ($(\overline{\mathsf{E}}[\mathsf{\Gamma}_i,\mathsf{E}]^+)_{i\in[\mu]}$ or \$) in a with replacement fashion, we sample the IVs in a without replacement manner, i.e., all the IVs will be distinct. Let the appropriately modified oracles be $(\widetilde{\mathsf{E}}[\mathsf{\Gamma}_i,\mathsf{E}])_{i\in[\mu]}$ and $\widetilde{\$}$. This switching is possible at the cost of two times the statistical distance between with and without replacement samples of size q, i.e. $q^2/2^\tau$. Formally, we have

$$\begin{aligned}&\left|\Pr\left(\mathscr{A}'^{(\overline{\mathsf{E}}[\mathsf{\Gamma}_i,\mathsf{E}]^+)_{i\in[\mu]}}=1\right) - \Pr\left(\mathscr{A}'^{\$}=1\right)\right| \\ &\quad\le \left|\Pr\left(\mathscr{A}'^{(\overline{\mathsf{E}}[\mathsf{\Gamma}_i,\mathsf{E}]^+)_{i\in[\mu]}}=1\right) - \Pr\left(\mathscr{A}'^{(\widetilde{\mathsf{E}}[\mathsf{\Gamma}_i,\mathsf{E}]^+)_{i\in[\mu]}}=1\right)\right| \\ &\qquad + \left|\Pr\left(\mathscr{A}'^{(\widetilde{\mathsf{E}}[\mathsf{\Gamma}_i,\mathsf{E}]^+)_{i\in[\mu]}}=1\right) - \Pr\left(\mathscr{A}'^{\widetilde{\$}}=1\right)\right| \\ &\qquad + \left|\Pr\left(\mathscr{A}'^{\widetilde{\$}}=1\right) - \Pr\left(\mathscr{A}'^{\$}=1\right)\right| \\ &\quad\le \left|\Pr\left(\mathscr{A}'^{(\widetilde{\mathsf{E}}[\mathsf{\Gamma}_i,\mathsf{E}]^+)_{i\in[\mu]}}=1\right) - \Pr\left(\mathscr{A}'^{\widetilde{\$}}=1\right)\right| + \frac{q^2}{2^\tau}. \quad (4)\end{aligned}$$

Now, we define $\mathscr{D}$ as a $(q, 1, q, \ell, \sigma, \dot{t})$-priv\$ distinguisher that runs $\mathscr{A}'$ in a black box manner. For each query (m, i) from $\mathscr{A}'$, $\mathscr{D}$ chooses a fresh user ID t from the set $\{0,1\}^\tau$ in a without

replacement manner. It then queries (m,t) to its own oracle (either $(\mathsf{E}^{+}_{\Gamma_i(t)}(m))_{i\in[\mu]}$ or $(\overline{\Gamma}_j(t,m))_{j\in[q]}$), where $(\overline{\Gamma}_j)_{j\in[q]} \hookleftarrow \mathcal{F}_{\mathsf{lp}}(\{0,1\}^\tau \times \mathcal{M}, \mathcal{M})$, and returns (t,c) to $\mathscr{A}'$, where c is the corresponding response of $\mathscr{D}$'s oracle. At the end $\mathscr{D}$ outputs the same value as $\mathscr{A}'$. It is obvious to see that $\mathscr{D}$ correctly simulates $\widetilde{\$}$ when it is interacting with $(\overline{\Gamma}_j)_{j\in[q]}$. Also, since Γ is a random function, $\mathscr{D}$ correctly simulates $(\widetilde{\mathsf{E}}[\Gamma_i,\mathsf{E}]^{+})_{i\in[\mu]}$ when it is interacting with $(\mathsf{E}^{+}_{\Gamma_i(\cdot)})_{i\in[\mu]}$. Moreover, $\mathscr{D}$ makes at most q queries to its oracle and runs in time $\dot{t} = O(t + qt_{\mathsf{E}})$. Thus, we have

$$\begin{aligned}\mathbf{Adv}^{\mathsf{priv\$}}_{\mathsf{E}}(\mathscr{D}) &= \left| \Pr_{\substack{(\Gamma_i)_{i\in[\mu]} \\ \hookleftarrow \mathcal{F}(\tau,\kappa+n)}} \left(\mathscr{D}^{(\mathsf{E}^{+}_{\Gamma_i(\cdot)})_{i\in[\mu]}} = 1\right) - \Pr_{\substack{(\overline{\Gamma}_j)_{j\in[q]} \hookleftarrow \$ \\ \mathcal{F}_{\mathsf{lp}}(\{0,1\}^\tau\times\mathcal{M},\mathcal{M})}} \left(\mathscr{D}^{(\overline{\Gamma}_j)_{j\in[q]}} = 1\right) \right| \\ &\geq \left| \Pr\left(\mathscr{A}'^{(\widetilde{\mathsf{E}}[\Gamma_i,\mathsf{E}]^{+})_{i\in[\mu]}} = 1\right) - \Pr\left(\mathscr{A}'^{\widetilde{\$}} = 1\right) \right|. \end{aligned} \tag{5}$$

The result follows from Eq. (2)-(5). □

3.2 MTProto2.0

In MTProto2.0, the three underlying functions, F, G, and E are constructed using the hash function SHA-256 [NIS15] and the IV-based encryption mode of operation Infinite Garble Extension [Cam78] or IGE.

SHA-256: It uses Merkle-Damgård paradigm [Mer89, Dam89] with a Davies-Meyer compression function [PGV93] and length-strengthened padding. A simplified version of SHA-256 algorithm is illustrated in Figure 3.2. Let $r = 512$, $c = 256$, $\ell = 64$, and $f \in \mathcal{F}(r+c, c)$. We define the length-strengthened padding function $\mathsf{pad}_r : \{0,1\}^{<2^\ell} \to \{0,1\}^{r+}$, where $\{0,1\}^{r+}$ (resp. $\{0,1\}^{<2^\ell}$) denotes the set of (non-empty) bit strings whose length is a multiple of r (resp. of length smaller than 2^ℓ), by the mapping

$$m \mapsto m\|10^d\|\langle|m|\rangle_\ell,$$

where $d = \min\{i \geq 0 \ : \ |m|+1+i+\ell \pmod r \equiv 0\}$ and $\langle|m|\rangle_\ell$ denotes the 64-bit unsigned binary representation of $|m|$. Let $\mathrm{iv} \in \{0,1\}^c$ be some application constant. Formally, the SHA-256 algorithm based on compression function f is defined as follows:

> for all $m \in \{0,1\}^{<2^\ell}$, we write $(m_1,\ldots,m_l) \coloneqq \mathsf{pad}_r(m)$, $h_0 \coloneqq \mathrm{iv}$,
>
> $$h_i \coloneqq f(m_i, h_{i-1}) \oplus h_{i-1}, \qquad 1 \leq i \leq l,$$
>
> and finally, $\mathsf{SHA\text{-}256}(m) \coloneqq h_l$. We refer to h_i values as *compression input* and the r-bit inputs as *compression key.*

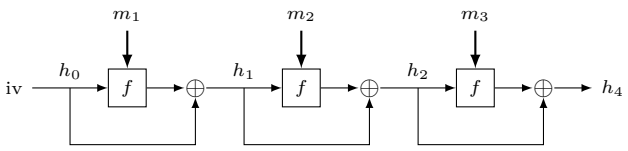

Figure 3.2: SHA-256 hash computation over a 3-block padded message $m_1\|m_2\|m_3 = \mathsf{pad}_r(m)$.

Note that the mapping $(m_i, h_{i-1}) \mapsto h_i$ applies the well-known Davies-Meyer transformation using f as the underlying primitive. In MTProto2.0, SHA-256 is exclusively used to construct two hash based PRFs F and G. Before we describe these functions, we first digress a little to discuss the security assumption on f vis-à-vis the security analysis of F and G.

Security Assumption on f: It is worth noting that, f can actually be viewed as a block cipher with an r-bit key and a c-bit block. Indeed, later we assume that the underlying block cipher in SHA-256, i.e. the f function, is a *tweakey block cipher* — a tweakable block cipher following the TWEAKEY framework by Jean et al. [JNP14]. Among other things, this framework allows an intermixing of tweaks and keys as a single input called tweakey. In context of our requirement, this can be looked in another way, where f, a tweakey block cipher with key space $\{0,1\}^{\frac{\kappa}{2}}$, tweak space $\{0,1\}^{r-\frac{\kappa}{2}}$, and block space $\{0,1\}^c$, can actually be seen as a tweakable block cipher with key space $\{0,1\}^{\frac{\kappa}{2}}$, tweak space $\mathcal{B} \times \{0,1\}^{r-\frac{\kappa}{2}}$, and block space $\{0,1\}^c$, where $\mathcal{B}$ is a subset of all r-bit strings having $(r - \frac{\kappa}{2})$ hamming weight. Further, each $b = b_0 \| \cdots \| b_{r-1} \in \mathcal{B}$ pinpoints the placement of tweak bits ($t \in \{0,1\}^{r-\frac{\kappa}{2}}$) in the actual tweakey. For $i = 0$ to $r-1$, if $b_i = 1$, then the i-th bit of the tweakey holds the next-in-line (starting from the first) bit of tweak t. While there is no explicit analysis of this feature, the framework itself does not distinguish between key and tweak, or their respective placement in the tweakey. Each choice of placement gives a new TBC, which justifies our assumption. For example, as per the Deoxys v1.43 specification [JNPS18, Section 6.2] or the corresponding journal publication [JNPS21, Section 5.3], the Deoxys-BC-256 block cipher can be made tweakable by announcing some bits as tweak. Those bits are usually chosen to maximize performance, but their position should have no impact on security as long as keys are uniformly random and independent. Indeed practical examples like Skinny and Mantis [BJK+16] also satisfy this criteria, i.e., one can choose multiple b vectors freely to create multiple instances.

Specifically, in case of f, we define $\mathcal{B} = \{B_a \coloneqq 1^\tau \| 0^{\frac{\kappa}{2}} \| 1^{r-\frac{\kappa}{2}-\tau} \| 1^{r-\frac{\kappa}{2}}, B_b \coloneqq 0^{\frac{\kappa}{2}} \| 1^{r-\frac{\kappa}{2}}\}$.

G function: For simplicity we assume a one-way communication from clients to the server. Let $\kappa = 576$ and $\tau = 128$. The G function takes a κ-bit key $k = (k_0, k_1)$, where $|k_b| = \kappa/2$, and a τ-bit input x and produces a $2c$-bit output $y = (y_0, y_1)$, where $|y_b| = c$. Internally, G can be viewed as two parallel invocations of SHA-256 with independent keys. Formally, for key (k_0, k_1) and input t, we have $y = \mathsf{G}_k(x) \coloneqq (\overline{\mathsf{G}}_{k_0}(x), \widetilde{\mathsf{G}}_{k_1}(x))$, where

$$\overline{\mathsf{G}}_{k_0}(x) \coloneqq a[0,\ldots,7]||b[8,\ldots,23]||a[24,\ldots,31]$$
$$\widetilde{\mathsf{G}}_{k_1}(x) \coloneqq b[0,\ldots,7]||a[8,\ldots,23]||b[24,\ldots,31]$$

and $a = \mathsf{SHA\text{-}256}(x\|k_0), b = \mathsf{SHA\text{-}256}(k_1\|x)$. In Lemma 3.2, under the assumption that f is a secure tweakey block cipher, we show that G is a secure weak PRF as is required in Lemma 3.1.

Lemma 3.2. *Let D be a (μ, q, t) wPRF distinguisher against G that issues at most q queries over at most μ users, and runs in time at most t. Then there exists a $(\mu, 2q, t')$ TPRP distinguisher D' against f such that we have*

$$\mathbf{Adv}^{\mathsf{wprf}}_{\mathsf{G}}(D) \leq \mathbf{Adv}^{\mathsf{tprp}}_{f}(D')$$

Proof. First, using the tweakey block cipher description of f, we can redefine a and b as:

$$a \coloneqq f_{k_0}((B_a, x\|10^{31}\|\langle\tau\rangle_{64}), \mathrm{iv}) \oplus \mathrm{iv}$$
$$b \coloneqq f_{k_1}((B_b, x\|10^{31}\|\langle\tau\rangle_{64}), \mathrm{iv}) \oplus \mathrm{iv}$$

where $B_a = 1^\tau\|0^{\frac{\kappa}{2}}\|1^{r-\frac{\kappa}{2}-\tau}$ and $B_b \coloneqq 0^{\frac{\kappa}{2}}\|1^{r-\frac{\kappa}{2}}$. Now, using a simple hybrid argument we can replace all the instances of f with tweakable random permutations, which incur a cost of at most $\mathbf{Adv}^{\mathsf{tprp}}_{f}(D')$. The remainder of the proof follows from the fact that the output distribution of a tweakable random permutation is identical to a random function, given that it is always invoked with distinct tweaks. □

$\mathrm{Re}^{\mathsf{prfleak}}_{\mathsf{DMD},\mathsf{chop}_\tau}(D)$ | $\boxed{\mathrm{Id}^{\mathsf{prfleak}}_{\mathsf{DMD},\mathsf{chop}_\tau}(D)}$

$u \leftarrow 0$
$b \leftarrow D^{\textsc{New},\mathsf{O}}$
return $b =_? 1$

Oracle $\textsc{New}()$

$u \leftarrow u + 1$
$K_u \leftarrow\!\!\$\ \{0,1\}^c$
$\boxed{\Gamma_u \leftarrow\!\!\$\ \mathcal{F}(\{0,1\}^r, \{0,1\}^c)}$
return $\mathsf{chop}_\tau(K_u)$

Oracle $\mathsf{O}(x, i)$

if $i \notin [u]$ **then**
  return $\perp$
$y \leftarrow \mathsf{DMD}_f(K_i, x)$
$\boxed{y \leftarrow \Gamma_i(x)}$
return y

Figure 3.3: Pseudorandom function under leakage game.

F FUNCTION: This function is simply defined as $\mathsf{F}_k(m) \coloneqq \mathsf{chop}_\tau(\mathsf{SHA\text{-}256}(k\|m))$ for all keys $k \in \{0,1\}^{r/2}$ and messages $m \in \{0,1\}^\ell$, where $\mathsf{chop}_\tau(\cdot)$ returns a substring of its output of length τ bits. In essence, F is nothing but the popular hash-based MAC construction called AMAC [BDL+11, BBT16]. In [BBT16], Bellare et al. showed that AMAC is a multi-user secure PRF under the assumption that the underlying compression function is a secure PRF under the presence of leakage. In the same paper, they also show that under reasonable theoretical assumptions (ideal cipher model), a Davies-Meyer style compression function is indeed secure PRF under the presence of leakage via truncation. Here, we restate their result in our setting and for F. First, we define two keyed functions $\mathsf{DM}[f] : \{0,1\}^r \times \{0,1\}^c \to \{0,1\}^c$ and $\mathsf{DMD}[f] : \{0,1\}^c \times \{0,1\}^r \to \{0,1\}^c$ with r-bit and c-bit keys respectively, as follows

$$\mathsf{DM}_f(k, x) \coloneqq f_k(x) \oplus x$$
$$\mathsf{DMD}_f(k, x) \coloneqq \mathsf{DM}_f(x, k).$$

Pseudorandom Function under Leakage: We use the PRF security under leakage notion by Bellare et al. [BBT16] described via the game in Figure 3.3. The PRF under leakage advantage of any adversary D against DMD and leakage function chop_τ is defined as

$$\mathbf{Adv}^{\mathsf{prfleak}}_{\mathsf{DMD}_f,\mathsf{chop}_\tau}(D) \coloneqq \left|\Pr\left(\mathrm{Re}^{\mathsf{prfleak}}_{\mathsf{DMD}_f,\mathsf{chop}_\tau}(D) = 1\right) - \Pr\left(\mathrm{Id}^{\mathsf{prfleak}}_{\mathsf{DMD}_f,\mathsf{chop}_\tau}(D) = 1\right)\right|.$$

Lemma 3.3 (Theorem 5.3 in [BBT16])**.** *Let $\mathscr{D}$ be a $(\mu, q, \ell, \sigma, t)$ PRF distinguisher against F, that issues at most q queries of length at most ℓ n-bit blocks, over at most u users, for a total queries length of at most σ n-bit blocks. Then, there exists (μ, q, t'') and $(\mu, q, \ell, \sigma, t')$ distinguishers $\mathscr{A}$ and $\mathscr{B}$, respectively, such that*

$$\mathbf{Adv}^{\mathsf{prf}}_{\mathsf{F}}(\mathscr{D}) \leq 2\mathbf{Adv}^{\mathsf{prf}}_{DM_f}(\mathscr{A}) + \ell\mathbf{Adv}^{\mathsf{prfleak}}_{DMD_f,\mathsf{chop}_\tau}(\mathscr{B}).$$

INFINITE GARBLE EXTENSION: The mode is an extension of CBC encryption [EMST76]. Basically, in addition to the CBC like ciphertext feed forward to the next block cipher input, this mode also employs plaintext feed forward to the next block cipher output. It is illustrated in Figure 3.4.

Formally, we define the IGE construction as follows: for a positive integer n and a key $k \in \mathcal{K}$, let E_k be a permutation of $\{0,1\}^n$. Let $\mathcal{I} = \{0,1\}^{2n}$ be the nonce space and $\mathcal{M} = \{0,1\}^{\ell n}$ for some integer ℓ be the message space. For every $m = (m_1, \ldots, m_\ell) \in \mathcal{M}$ and $(\mathrm{iv}_1, \mathrm{iv}_2) \in \mathcal{I}$ the encryption function is defined as,

$$\mathsf{E}^+_{k,\mathrm{iv}_1,\mathrm{iv}_2}(m) = c = (c_1, \ldots, c_\ell),$$

where

$$c_i = \begin{cases} E_k(\mathrm{iv}_1 \oplus m_1) \oplus \mathrm{iv}_2, & i = 1 \\ E_k(c_{i-1} \oplus m_i) \oplus m_{i-1}, & i > 1 \end{cases}.$$

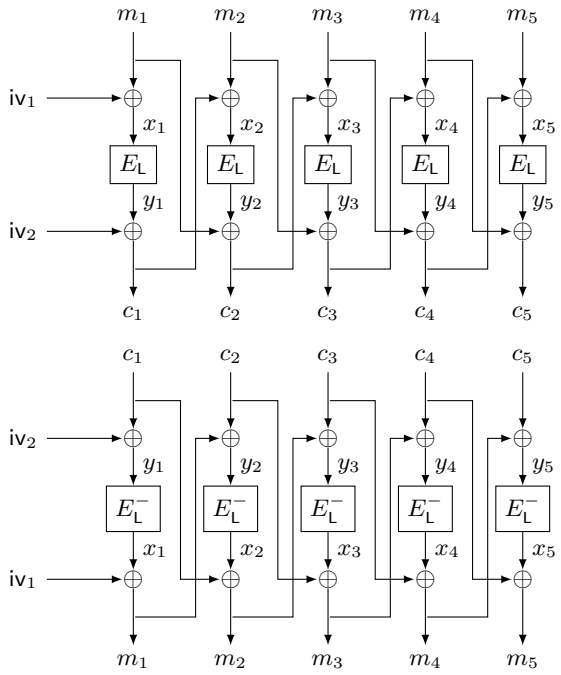

Figure 3.4: IGE encryption (**top**) and decryption (**bottom**) algorithms.

Similarly, we define the decryption function as follows. For every $c = (c_1, \ldots, c_\ell) \in \mathcal{M}$ and $(\mathrm{iv}_1, \mathrm{iv}_2) \in \mathcal{I}$,

$$\mathsf{E}^-_{k,\mathrm{iv}_1,\mathrm{iv}_2}(c) = m = (m_1, \ldots, m_\ell),$$

where

$$m_i = \begin{cases} E_k^{-1}(\mathrm{iv}_2 \oplus c_1) \oplus \mathrm{iv}_1, & i = 1 \\ E_k^{-1}(m_{i-1} \oplus c_i) \oplus c_{i-1}, & i > 1 \end{cases}.$$

For most part of our analysis, only the privacy security of IGE will suffice. Accordingly, we bound the advantage of a distinguisher that tries to distinguish between IGE and a uniform random string generator in Lemma 3.4.

Lemma 3.4. *Let A be a $(\mu, q_{\max}, q, \ell, \sigma, t)$ multi-user distinguisher against* IGE*, that runs in time at most t, and such that each user $u \in [\mu]$ makes q_u queries (the maximal number of queries for a single user is denoted by q_{max}), each of length (in n-bit blocks) at most ℓ, of total queries length at most σ. Then there exists a multi-user distinguisher A' against* PRP *with u users, at most $q_u \cdot \ell$ queries per user, and total σ queries across all users such that,*

$$\mathbf{Adv}^{\mathsf{priv\$}}_{\mathsf{IGE}}(A) \leq \mathbf{Adv}^{\mathsf{prp}}_{E}(A') + \frac{2\mu q_{max}\ell^2}{2^n}.$$

Proof. First, using a simple straightforward hybrid argument, for our adversary A there exists a multi-user distinguisher A' against PRP as described above such that,

$$\mathbf{Adv}^{\mathsf{priv\$}}_{\mathsf{IGE}}(A) \leq \mathbf{Adv}^{\mathsf{prp}}_{E}(A') + \delta(A),$$

where $\delta(A)$ is the advantage of A against IGE where the permutation for each user E_{k_u} is replaced by a uniform random permutation of $\{0,1\}^n$. Note that all these random permutations are independent from one another.

For the remainder of this proof we employ the Coefficient-H technique by Patarin [Pat91, Pat08], which is a tool to upper bound the distinguishing advantage of any

deterministic and computationally unbounded distinguisher $\mathscr{A}$ in distinguishing the real oracle $\mathcal{R}$ from the ideal oracle $\mathcal{I}$. The collection of all queries and responses that $\mathscr{A}$ made and received to and from the oracle, is called the transcript of $\mathscr{A}$, denoted as τ.

Let $\mathbb{T}_{\mathsf{re}}$ and $\mathbb{T}_{\mathsf{id}}$ denote the transcript random variable induced by $\mathscr{A}$'s interaction with $\mathcal{R}$ and $\mathcal{I}$, respectively. Let $\mathcal{T}$ be the set of all transcripts. A transcript $\tau \in \mathcal{T}$ is said to be *attainable* if $\Pr(\mathbb{T}_{\mathsf{id}} = \tau) > 0$, i.e., it can be realized by $\mathscr{A}$'s interaction with $\mathcal{I}$. Following these notations, we state the main result of coefficient-H technique in Theorem 3.1. A proof of this theorem is available in [CS14, MN17], among others.

Theorem 3.1. *For $\epsilon_1, \epsilon_2 \geq 0$, suppose there is a set $\mathcal{T}_{\mathsf{bad}} \subseteq \mathcal{T}$, that we call the set of bad transcripts, such that the following conditions hold:*

- $\Pr(\mathbb{T}_{\mathsf{id}} \in \mathcal{T}_{\mathsf{bad}}) \leq \epsilon_1$*; and*
- *For any $\tau \notin \mathcal{T}_{\mathsf{bad}}$, τ is attainable and* $\dfrac{\Pr(\mathbb{T}_{\mathsf{re}} = \tau)}{\Pr(\mathbb{T}_{\mathsf{id}} = \tau)} \geq 1 - \epsilon_2$.

Then, for any computationally unbounded and deterministic distinguisher $\mathscr{A}$, we have

$$\mathbf{Adv}_{\mathcal{R};\mathcal{I}}(\mathscr{A}) \leq \epsilon_1 + \epsilon_2.$$

In our scenario, A prompts a user $u \in [\mu]$ and makes a query to one of two oracles. The *Real Oracle* is the IGE construction (where the key used is k_u) while the *Ideal Oracle* is a function $\$_u(\cdot)$ that takes a message $m \in \mathcal{M}$ and returns a uniform random string of size $n(\ell+2)$ that consists of a pair $(\mathrm{iv}_1^u, \mathrm{iv}_2^u) \in \mathcal{I}$ and a cipher text $c \in \mathcal{M}$. In conclusion, the distinguisher A has access to the following transcript,

$$\tau(A) = \{(m^{i,u}, c^{i,u}, \mathrm{iv}_1^{i,u}, \mathrm{iv}_2^{i,u}) : u \in [\mu], i \in [q_u]\},$$

where for every $i \in [q_u]$, $m^{i,u}, c^{i,u} \in \mathcal{M}$ and $\mathrm{iv} = (\mathrm{iv}_{i,1}, \mathrm{iv}_{i,2}) \in \mathcal{I}$ are chosen uniformly and independently from $\mathcal{I}$.

In order to upper bound the advantage of A over IGE we will need some notations. Using the transcripts above, one can define the following intermediate values. For every user $u \in [\mu]$, $i \in [q_u], j \in [\ell]$ let

$$x_{i,j}^u = \begin{cases} m_1^{i,u} \oplus \mathrm{iv}_1^{i,u}, & j = 1 \\ m_j^{i,u} \oplus c_{j-1}^{i,u}, & j > 1 \end{cases}, \quad y_{i,j}^u = \begin{cases} c_1^{i,u} \oplus \mathrm{iv}_2^{i,u}, & j = 1, \\ c_j^{i,u} \oplus m_{j-1}^{i,u}, & j > 1 \end{cases}.$$

Notice that if A prompts user $u \in [\mu]$ and gets an answer from the *Real Oracle*, then for every $(i,j) \neq (i',j') \in [q_u] \times [\ell]$, $x_{i,j}^u = x_{i',j'}^u \Leftrightarrow y_{i,j}^u = y_{i',j'}^u$. The property holds because $y_{i,j}^u = E_{k_u}(x_{i,j}^u)$, $y_{i',j'}^u = E_{k_u}(x_{i',j'}^u)$ and E_{k_u} is a permutation. Therefore when A receives a transcript where this property does not hold he gains advantage. To avoid such scenarios, we say a transcript τ is bad if one of the following conditions hold,

$$\exists u \in [\mu], (i,j) \neq (i',j') \in [q_u] \times [\ell] : x_{i,j}^u = x_{i',j'}^u, y_{i,j}^u \neq y_{i',j'}^u, \tag{B1}$$

$$\exists u \in [\mu], (i,j) \neq (i',j') \in [q_u] \times [\ell] : y_{i,j}^u = y_{i',j'}^u, x_{i,j}^u \neq x_{i',j'}^u. \tag{B2}$$

Finally, we denote by T_{BAD} the set of all possible bad transcripts and by T_{GOOD} the rest.

Next, notice that the probability that a transcript is bad is non zero only if we invoke the *Ideal Oracle*. Hence, by the definition of $\$_u(\cdot)$, for every $(i,j) \neq (i',j')$ the random variables $c_j^{i,u}$ and $c_{j'}^{i',u}$ are independent uniform random variables over $\{0,1\}^n$. Then, one has,

$$\Pr(x_{i,j}^u = x_{i',j'}^u) = \Pr\left(c_{j-1}^{i,u} \oplus c_{j'-1}^{i',u} = m_j^{i,u} \oplus m_{j'}^{i',u}\right) = \frac{1}{2^n}.$$

Similarly,

$$\Pr(y^u_{i,j} = y^u_{i',j'}) = \frac{1}{2^n}$$

In conclusion,

$$\begin{aligned}\Pr(\tau(A) \in T_{BAD}) &= \Pr(\tau(A) \text{ satisfy B1}) + \Pr(\tau(A) \text{ satisfy B2}) \\ &\leq \sum_{u\in[\mu]} \sum_{(i,j)\neq(i',j')\in[q_u]\times[\ell]} \Pr(x^u_{i,j} = x^u_{i',j'}, y^u_{i,j} \neq y^u_{i',j'}) \\ &+ \Pr(y^u_{i,j} = y^u_{i',j'}, x^u_{i,j} \neq x^u_{i',j'}) \leq \frac{2\mu q_{max}\ell^2}{2^n}.\end{aligned}$$

Finally, we analyze the probability to encounter a given good transcript $\tau(A)$. For the *Real Oracle* one has,

$$\Pr\left(\mathbb{T}_{\mathsf{re}} = \tau(A)\right) = \prod_{u=1}^{\mu} \frac{1}{2^{2q_u n} \cdot (2^n)_{q_u\ell}}.$$

While for the *Ideal Oracle*,

$$\Pr\left(\mathbb{T}_{\mathsf{id}} = \tau(A)\right) = \prod_{u=1}^{\mu} \frac{1}{2^{q_u n(\ell+2)}}.$$

It is easy to see that for every user $u \in [\mu]$,

$$\frac{2^{q_u n\ell}}{(2^n)_{q_u\ell}} \geq 1$$

Hence,

$$\frac{\Pr\left(\mathbb{T}_{\mathsf{re}} = \tau\right)}{\Pr\left(\mathbb{T}_{\mathsf{id}} = \tau\right)} = \prod_{u=1}^{\mu} \frac{2^{q_u n\ell}}{(2^n)_{q_u\ell}} \geq 1$$

In conclusion, according to Theorem 3.1,

$$\mathbf{Adv}^{\mathsf{priv\$}}_{\mathsf{IGE}}(A) \leq \mathbf{Adv}^{\mathsf{prp}}_{E}(A') + \frac{2\ell^2\mu q_{max}\ell^2}{2^n}.$$

□

Note that, in context of MTProto-G, for each message a uniform at random (assuming KDF is PRF) key is chosen from the key space. Therefore, $q_{\max}$ is essentially 1. Also, IGE is slightly stronger as compared to the usual IV-based encryption used in SIV, i.e., the CTR mode. In case of IGE even if two IVs collides, there is still some security as long as the first block of the corresponding messages are different. This does not hold for CTR mode. Now combining the results in Lemma 3.1, 3.2, 3.3, and 3.4 we get the following security bound for MTProto2.0.

Corollary 3.1. *For $\mu, q, \ell, \sigma, t > 0$, the DAE security of MTProto2.0 is given by*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{dae}}_{\mathit{MTProto2.0}}(\mathscr{A}) \leq 2\mathbf{Adv}^{\mathsf{prf}}_{\mathit{DM}_f}(\mathscr{B}) + \ell\mathbf{Adv}^{\mathsf{prfleak}}_{\mathit{DMD}_f,\mathsf{chop}_\tau}(\mathscr{B}') + \mathbf{Adv}^{\mathsf{tprp}}_{f}(\mathscr{C}) \\ + \mathbf{Adv}^{\mathsf{prp}}_{E}(\mathscr{D}) + \frac{2\mu q_{max}\ell^2}{2^n} + \frac{q}{2^\tau} + \frac{q^2}{2^\tau}\end{aligned}$$

4 Presentation of the Full MTProto2.0 Protocol

4.1 Client-Server Encrypted Communication

Encoding of a Message. Like in MTProto 1.0, the DH key exchange is used to generate a shared key between the sender and the receiver. After the key exchange, the sender and the receiver share a 2048-bit symmetric key denoted by K and an additional key fingerprint f defined as the last 64 bits of SHA-1 on K. This fingerprint is used as a sanity check for the key exchange procedure to detect bugs in the software implementation. Moreover, in order to keep past communications safe, the secret key is regenerated once a key has been used for more than 100 messages or more than a week.
Next, we define the *protocol enriched message* X as:

$$\mathsf{X} := \mathbf{salt}\|\mathbf{session\_id}\|\mathbf{message\_id}\|\mathbf{seq\_no}\|\mathbf{message\_data\_length}\|\mathbf{message\_data}, \tag{6}$$

where

- **salt**(64-bit) - Changed every 30 minutes (separately for each session) at the request of the server. All subsequent messages must contain the new salt (although, messages with the old salt are still accepted for a further 30 minutes). Required to protect against replay attacks and certain tricks associated with adjusting the client clock to a moment in the distant future.
- **session_id**(64-bit) - Generated by the client to distinguish between individual sessions (for example, between different instances of the application, created with the same authorization key). The session in conjunction with the key identifier corresponds to an application instance.
- **message_id**(64-bit) - Time-dependent number used uniquely to identify a message within a session. Client message identifiers are divisible by 4, server message identifiers modulo 4 yield 1 if the message is a response to a client message, and 3 otherwise. Client message identifiers must increase monotonically (within a single session), the same as server message identifiers, and must approximately equal $unixtime * 2^{32}$. This way, a message identifier points to the approximate moment in time the message was created. A message is rejected over 300 seconds after it is created or 30 seconds before it is created (this is needed to protect from replay attacks). In this situation, it must be re-sent with a different identifier (or placed in a container with a higher identifier). The identifier of a message container must be strictly greater than those of its nested messages.
- **seq_no**(32-bit) - Equal to twice the number of "content-related" messages (those requiring acknowledgment, and in particular those that are not containers) created by the sender prior to this message and subsequently incremented by one if the current message is a content-related message. A container is always generated after its entire contents; therefore, its sequence number is greater than or equal to the sequence numbers of the messages contained in it.
- **message_data_length**(32-bit).
- **message_data**.

In addition, we also have the random padding value, **random_padding**, that consists of 12 to 1024 bytes to make its length divisible by 16 bytes (the sampling is described in detail in Section 4.1). Finally, we define the *fully encoded message* for encryption X' as

$$\mathsf{X}' := \mathsf{X}\|\mathbf{random\_padding} \tag{7}$$

Let y be a chat parameter defined as $y = 0$ for messages from client to server and $y = 8$ otherwise. We define the authentication tag t, first let $k_1 = K[88 + y, \ldots, 119 + y]$ be some middle bytes of the shared key then we define $t = \mathsf{F}_{k_1}(\mathsf{X}')$. Notice that t is also used for deriving keys in the encryption of E. In order to use the encryption scheme E, we first generate the key and iv for the encryption scheme. For that let $k_2 = K[y, \ldots, 35 + y], k_3 = K[40 + y, \ldots, 75 + y]$ be some part of the shared key then for $k = (k_2, k_3)$ we have that $\mathsf{G}_k(t) = (l, \mathrm{iv})$.

Finally, the encryption scheme (uses AES-256 with IGE mode) is defined as, $\mathsf{E}^+_{l,\mathrm{iv}}(\mathsf{X}') = c$, where the final cipher-text returned is defined as the string, $(f||t||c)$.

Sampling of a Random Padding. The padding algorithm of MTProto2.0 is uncommon, and deserves to be described in more details. Indeed, the padding is filled with random bits, and its length is also chosen at random[2]. It is worth noting that each official client seems to use a different algorithm to randomize padding length. Since our goal is to focus on building a hypothetical malicious client, we will present the two length randomization algorithms that are best suited to our attack. We start with the algorithm of the desktop client. Let us assume that we want to encrypt a σ-byte message M. Let us write $\sigma = 16q + 4r + s$, where $0 \leq r < 4$, and $0 \leq s < 4$. The padded message will consist $\sigma' = 4q + r + f(r) + 4\mathbf{Rand}(0, 15)$ 32-bit blocks, where $f(0) = 4$, $f(1) = 3$ $f(2) = 6$, $f(3) = 5$, and $\mathbf{Rand}(0, 15)$ denotes an integer that is chosen uniformly at random between 0 and 15. In particular, this means that, after padding, the length of the plaintext in 32-bit blocks will be

$$\sigma' = 4\left(q + \lfloor r/2 \rfloor + 1 + \mathbf{Rand}(0, 15)\right).$$

Let us define the function

$$\begin{aligned} g : \mathbb{N} &\longrightarrow \mathbb{N} \\ 16q + 4r + s &\longmapsto q + \lfloor r/2 \rfloor + 1. \end{aligned}$$

Then, one clearly has $\sigma' = 4(g(\sigma) + \mathbf{Rand}(0, 15))$, and thus the size of the padded data in 128-bit blocks will be $\sigma_{128} = g(\sigma) + \mathbf{Rand}(0, 15)$. In particular, it means that $\sigma_{128} \mod 16$ is uniformly random. An equivalent way of sampling this value would be to generate a random integer $v \in \{0, \ldots, 15\}$, to write $g(\sigma) = 16q' + r'$, and then to choose $\sigma_{128} = 16q' + v$ if $v \geq r'$, or $\sigma_{128} = 16(q' + 1) + v$ otherwise. This second sampling mechanism will prove useful in the following section. We will denote this alternative padding rule $\mathsf{pad}(M, v)$, where M is padded to a message whose length in 16-byte blocks is equal to v modulo 16.

Similarly, we discuss the length randomization algorithm from the tdlib library, that can be used to develop third-party clients. Let σ be the byte-length of the message to be encrypted. The length of the padded message will be

$$\sigma' = 16 \times \left\lfloor \frac{\sigma + 27 + \mathbf{Rand}(0, 255)}{16} \right\rfloor.$$

Like in the previous case, $\sigma'/16 \mod 16$ will be uniformly random, and we can similarly define a reverse padding mechanism with the exact same probability distribution, where we generate the target $\sigma'/16 \mod 16$ value ℓ uniformly at random, and then the padded message length as

$$\sigma' = 16 \times \left\lfloor \frac{\sigma + 27 + (\ell \times 16 + \mathbf{Rand}(0, 15) - \sigma - 27 \mod 256)}{16} \right\rfloor.$$

[2] Although it seems optional in the source code of Telegram, we focus on the randomized-length padding scheme, as the official documentation presents this one.

4.2 End to End Encrypted Communication Protocol

Encoding of a Message. The encoding of a message is almost identical to the one in Section 4.1. The difference lies in the definition of protocol enriched message X (see Eq. (6)) and the chat parameter (y is equal to 0 if the current user is the chat creator and otherwise $y = 8$). In this setting, X is defined as

$$\mathsf{X} := \mathbf{length}\|\mathbf{payload\_type}\|\mathbf{random\_bytes}\|\mathbf{layer}\|\mathbf{in\_seq\_no} \\ \|\mathbf{out\_seq\_no}\|\mathbf{message\_type}\|\mathbf{message\_data}, \tag{8}$$

The auxiliary information can be summarized using the following fields.

- **length** (32-bit): Length of the payload.
- **payload_type** (32-bit).
- **random_bytes** ($\geq$ 128-bit): Set of random bytes to prevent content recognition in short encrypted messages. Clients are required to check that there are at least 15 random bytes included in each message. Messages with less than 15 random bytes must be ignored.
- **layer** (32-bit): Layer number.
- **in_seq_no** (32-bit): Twice the number of messages in the sender's inbox (including deleted and service messages), incremented by 1 if current user was not the chat creator.
- **out_seq_no** (32-bit): Twice the number of messages in the recipient's inbox (including deleted and service messages), incremented by 1 if current user was the chat creator.
- **message_type** (32-bit).
- **message_data**.

Note that, the sequence numbers are especially important for our subversion attack presented in Section 5. Finally, the fully encoded message X' in secret chat setting is generated by appending a random padding in exactly the same fashion (see Eq. (7)) as in the client-server chat. Besides, some Telegram clients do check that the random padding is at least 12 bytes long (notably the iOS client).

5 Subverting Secret Chats in MTProto2.0

As previously discussed in Section 2.1, state reset is a simple countermeasure that can make stateful subversion attacks easy to detect. In the general case, it is thus important to design stateless subversion attacks. When attacking encryption schemes used in complex protocols, this requirement can sometimes be alleviated. For example, if the protocol maintains an internal counter that is given to the encryption scheme via associated data or plaintext, then a subversion attack can simply rely on this external counter in order to act as a stateful attack, even if it does not directly maintain its state. This is exactly what happens during secret chats, as the MTProto2.0 protocol relies on sequence counters in order to uniquely identify each message. In more details, from any encryption query, it is possible to extract a monotonically increasing counter that only depends on the number of encryption queries issued by the client, and the only time where these counters are reset

is during the rekeying of the authenticated encryption scheme. We can thus rely on this counter in order to mount a stateful substitution attack.[3]

For the remainder of this section, we focus on the MTProto authenticated encryption scheme (without associated data). Moreover, we fix a padding scheme that generates ciphertexts whose length in 128-bit blocks is uniformly distributed modulo 16.[4] In order to create a subversion attack, we have to find a way to exploit the randomization of the encryption scheme in order to exfiltrate key bits. As described in Section 4.1, although the random values used as input are encrypted and authenticated, part of the randomized length of the padding is still visible by an adversary. It is thus possible to modify Algorithm 2.1 in order to exploit these characteristics to transmit 4 bits of key material for each encrypted message, as seen in Algorithm 5.1.

Algorithm 5.1 Pseudocode of our subversion attack. Here σ denotes the internal counter of the MTProto protocol corresponding to key K that appears in the header of each message (it is not updated by the encryption algorithm), E is a length-preserving deterministic encryption scheme, $\mathcal{E}$ is the MTProto AE scheme, and pad denotes its padding algorithm, as presented in section 4.1.

function $\widetilde{\mathcal{E}^{+}}(K_A, K, M, \sigma)$
  $Y \leftarrow E(K_A, K)$
  if $\sigma \leq \lceil |K|/4 \rceil$ **then**
    $\text{len} \leftarrow Y[4\sigma, 4\sigma + 3]$
  else
    $\text{len} \leftarrow\!\!\$ \{0, \ldots, 15\}$
  end if
  $M \leftarrow \mathsf{pad}(M, \text{len})$
  $(C, T) \leftarrow \mathcal{E}^{+}(K, M)$
  return (C, T)
end function

function $\widetilde{\mathcal{E}^{\text{EXT}}}(K_A, \mathbf{C}, \sigma)$
  $r \leftarrow \lceil |K|/4 \rceil$
  $Y \leftarrow ||_{j=0}^{r} \langle |\mathbf{C}[j]|/16 \mod 16 \rangle_4$
  $Y \leftarrow \langle Y \rangle_{|K|}$
  $K \leftarrow E^{-}(K_A, Y)$
  return K
end function

It is clear that, as long as E is secure, $\widetilde{\mathcal{E}^{+}}$ is indistinguishable from $\mathcal{E}^{+}$ in the context of the execution of MTProto2.0, and that the key recovery always succeeds given a sufficient number of ciphertexts. Formally, one has the following result.

Theorem 5.1. *Let $q_k \geq \lceil |K|/4 \rceil$, and let D be an adversary against the strong detectability of $\widetilde{\mathcal{E}}$, as defined in Algorithm 5.1, that uses at most q queries to at most u users, for a total of at most l bits, and runs in time at most t. Then there exists a distinguisher D' against the security of E that uses at most u queries, and runs in time $t + O(l)$, such that:*

$$\mathbf{Adv}^{\mathsf{det}}_{\mathcal{E},\widetilde{\mathcal{E}}}(D) \leq \mathbf{Adv}^{\mathsf{wprf}}_{E}(D'),$$
$$\mathbf{Adv}^{\mathsf{kr}}_{\mathcal{M},q_k}(\widetilde{\mathcal{E}}) = 1.$$

A state actor that has subverted the encryption algorithm of a client, and has access to the server-side (encrypted and ordered) transcript of the conversation, will thus be able to recover 4 key bits per message sent between two rekeying steps with probability 1. Unfortunately, this is not sufficient to break the security of MTProto. Indeed, each key can only encrypt or decrypt at most 100 messages. This limits the number of key bits that can be targeted to a maximum of 400, and more realistically to around 200 bits (assuming each party sends around 50 messages each). Since MTProto keys are much longer, this is not

[3] Even though the state is maintained by the protocol and not by the encryption scheme, we still make it explicit in the pseudocode of our attacks.

[4] This is the case for the desktop client and the tdlib library.

sufficient to allow a realistic guess of the remaining bits. Fortunately, the only source of randomness that has been used for the subversion attack is the length of the padded data. It is still possible to exploit the random generation of the padding bits to mount a variant of the attack from Algorithm 2.2, thus sending more key bits, at a heavier computational cost. The pseudocode of this new attack can be found in Algorithm 5.2. Given the fact that the MTProto protocol is an authenticated encryption scheme in two passes, we have optimized our attack by applying the $F_{K'_A}$ function to the authentication tag instead of to the whole ciphertext. We stress that this also allows us to defer the encryption pass after the main loop of $\widetilde{\mathcal{E}^+}_{\delta,s}$ has terminated, and to only compute it once, even if the authentication step has been repeated s times.

Algorithm 5.2 Pseudocode of the updated subversion attack $\widetilde{\mathcal{E}^+}_{\delta,s}$. We use the same notation as in Algorithm 5.1, and F a PRF with range $\{0,1\}^\delta$. Ciphertexts are denoted (C,T) where T refers to the authentication tag.

function $\widetilde{\mathcal{E}^+}_{\delta,s}(K_A, K'_A, K, M, \sigma)$
  $Y \leftarrow E(K_A, K)$
  if $\sigma \leq \lceil |K|/(4+\delta) \rceil$ **then**
    $\text{len} \leftarrow Y[(4+\delta)\sigma, (4+\delta)\sigma+3]$
  else
    $\text{len} \leftarrow_\$ \{0,\ldots,15\}$
  end if
  $i \leftarrow 0$
  do
    $M' \leftarrow \mathsf{pad}(M, \text{len})$
    $T \leftarrow \mathsf{F}(K, M')$
    $X \leftarrow F_{K'_A}(T)$
    $b \leftarrow X =_? Y[(4+\delta)\sigma+3, (4+\delta)(\sigma+1)-1]$
    $i \leftarrow i+1$
  while (not b) and $(i < s)$
  $(L, \text{iv}) \leftarrow \mathsf{G}(K, T)$
  $C \leftarrow \mathsf{E}^+(L, \text{iv}, M')$
  return (C, T)
end function

function $\widetilde{\mathcal{E}^{\text{EXT}}}_{\delta,s}(K_A, \mathbf{C}, \sigma)$
  $Y \leftarrow \epsilon$
  for $i \in \{1,\ldots,|K|/(4+\delta)\}$ **do**
    $Y \leftarrow Y || \langle |\mathbf{C}[i]|/16 \mod 16 \rangle_4$
    $Y \leftarrow Y || F_{K'_A}(T)$
  end for
  $K \leftarrow E^-(K_A, \langle Y \rangle_{|K|})$
  return K
end function

In order to study this new attack, we follow [BJK15] and introduce the min-entropy $\mathbf{H}_\infty(\mathcal{E}.\text{Auth})$ of the (randomized) tag generation algorithm $\mathcal{E}.\text{Auth}$.[5] Formally, we define

$$2^{-\mathbf{H}_\infty(\mathcal{E}.\text{Auth})} = \max \Pr\left(\mathcal{E}.\text{Auth}_K(\mathsf{pad}(M, \text{len})) = T\right),$$

where the maximum is taken over all possible keys K, plaintexts M, tag values T, and padding lengths len, and the probability is taken over the uniformly random draw of the (at least 96) padding bits. Our results assume that $2^{-\mathbf{H}_\infty(\mathcal{E}.\text{Auth})}$ is small. One has the following result with respect to the detectability and key recovery success of this algorithm.

Theorem 5.2. *Let $q \geq \lceil |K_E|/(4+\delta) \rceil$, and let D be an adversary against the strong detectability of $\widetilde{\mathcal{E}}$, as defined in Algorithm 5.2, that uses at most q queries to at most u users, for a total of at most l bits, and runs in time at most t. Then there exists D_E, and D_F, such that*

[5] This corresponds to the sampling of the padding, and the computation of F.

- *D_E makes at most 1 query per user to E, u queries in total, and runs in time $t + O(sql)$;*
- *D_F makes at most sq queries per user to F, and runs in time $t + O(sql)$,*

and

$$\mathbf{Adv}^{\mathsf{det}}_{\mathcal{E},\widetilde{\mathcal{E}}_{\delta,s}}(D) \leq \mathbf{Adv}^{\mathsf{wprf}}_{E}(D_E) + \mathbf{Adv}^{\mathsf{prf}}_{F}(D_F) + \frac{u^2}{2|\mathcal{K}|} + q^2 s^2 2^{-\mathbf{H}_\infty(\mathcal{E}.\mathrm{Auth})-1}.$$

Further, there exists D'_F such that D'_F makes at most sq queries per user to F, and runs in time $t + O(sql)$, and

$$\mathbf{Adv}^{\mathsf{kr}}_{\mathcal{M},q}(\widetilde{\mathcal{E}}_{\delta,s}) \geq 1 - \mathbf{Adv}^{\mathsf{prf}}_{F}(D'_F) - q\left(1 - \frac{1}{2^\delta}\right)^s - q^2 s^2 2^{-\mathbf{H}_\infty(\mathcal{E}.\mathrm{Auth})-1}.$$

Proof. The proof is very similar to the proofs of [BJK15, Theorems 4.1 and 4.2]. We provide it in two parts for the sake of completeness.

5.1 Proving the Strong Undetectability of the Subversion Attack

First, we consider undetectability. We will employ a sequence of games as shown in Figure 5.1 and 5.2. One has

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{det}}_{\mathcal{E},\widetilde{\mathcal{E}}}(D) &\leq \left|\Pr\left(\mathrm{Sub}^{\mathrm{det}}_{\mathcal{E},\widetilde{\mathcal{E}}}(D)\right) - \Pr\left(\mathrm{Re}^{\mathrm{det}}_{\mathcal{E},\widetilde{\mathcal{E}}}(D)\right)\right| \\
&\leq \left|\Pr\left(\mathrm{Sub}^{\mathrm{det}}_{\mathcal{E},\widetilde{\mathcal{E}}}(D)\right) - \Pr\left(I_0\right)\right| \\
&\quad + \left|\Pr\left(I_0\right) - \Pr\left(I_1\right)\right| + \left|\Pr\left(I_1\right) - \Pr\left(I_2\right)\right| \\
&\quad + \left|\Pr\left(I_2\right) - \Pr\left(I_3\right)\right| \\
&\quad + \left|\Pr\left(I_3\right) - \Pr\left(\mathrm{Re}^{\mathrm{det}}_{\mathcal{E},\widetilde{\mathcal{E}}}(D)\right)\right|.
\end{aligned}$$

The only difference between the games $\mathrm{Sub}^{\mathrm{det}}_{\mathcal{E},\widetilde{\mathcal{E}}}(D)$ and I_0 is the fact that Y is sampled using E_{K_A} in $\mathrm{Sub}^{\mathrm{det}}_{\mathcal{E},\widetilde{\mathcal{E}}}(D)$, whereas it is sampled using a lazily-sampled random function in I_0. Thus one has

$$\left|\Pr\left(\mathrm{Sub}^{\mathrm{det}}_{\mathcal{E},\widetilde{\mathcal{E}}}(D)\right) - \Pr\left(I_0\right)\right| \leq \mathbf{Adv}^{\mathsf{wprf}}_{E}(D_E).$$

Games I_0 and I_1 are identical until there is a collision between the keys of two users. Hence we have

$$\left|\Pr\left(I_0\right) - \Pr\left(I_1\right)\right| \leq \Pr\left(I_1 \text{ sets } \mathsf{bad}\right) \leq \frac{u^2}{2|\mathcal{K}|}.$$

In Game I_2, $F_{K'_A}$ is replaced with a lazily sampled uniformly random function, which gives

$$\left|\Pr\left(I_1\right) - \Pr\left(I_2\right)\right| \leq \mathbf{Adv}^{\mathsf{prf}}_{F}(D_F).$$

Games I_2 and I_3 are identical until there is a tag collision. Thus

$$\left|\Pr\left(I_2\right) - \Pr\left(I_3\right)\right| \leq \Pr\left(I_3 \text{ sets } \mathsf{bad}\right)$$

$\underline{\mathrm{Sub}^{\mathrm{det}}_{\mathcal{E},\widetilde{\mathcal{E}}_{\delta,s}}(D)}$

$u \leftarrow 0$
$(K_A, K'_A) \leftarrow\!\!\$\ \widetilde{\mathcal{K}}$
$b \leftarrow D^{\textsc{New},\textsc{Enc}}()$
return $b = 1$

$\underline{\text{Oracle } \textsc{New}()}$

$u \leftarrow u + 1$
$\sigma_u \leftarrow 0$
$K_u \leftarrow\!\!\$\ \mathcal{K}$

$\underline{\text{Oracle } \textsc{Enc}(M, A, i)}$

if $i \notin [u]$ **then**
  return $\perp$
$Y \leftarrow E_{K_A}(K_i)$
if $\sigma_i \leq \lceil |K_i|/(4+\delta) \rceil$
  $\mathrm{len} \leftarrow Y[(4+\delta)\sigma_i,$
    $(4+\delta)\sigma_i + 3]$
else
  $\mathrm{len} \leftarrow\!\!\$\ \{0, \ldots, 15\}$
$j \leftarrow 0$
do
  $M' \leftarrow \mathsf{pad}(M, \mathrm{len})$
  $(C, T) \leftarrow \mathcal{E}^+_{K_i}(M')$
  $X \leftarrow F_{K'_A}(T)$
  $b \leftarrow X = Y[(4+\delta)\sigma_i + 3,$
    $(4+\delta+1)\sigma_i - 1]$
  $j \leftarrow j + 1$
while (not b) and $(j < s)$
$\sigma_i \leftarrow \sigma_i + 1$
return (C, T)

$\underline{\boxed{I_0}\ I_1}$

$u \leftarrow 0$
$(K_A, K'_A) \leftarrow\!\!\$\ \widetilde{\mathcal{K}}$
$S \leftarrow \emptyset$
$b \leftarrow D^{\textsc{New},\textsc{Enc}}()$
return $b = 1$

$\underline{\text{Oracle } \textsc{New}()}$

$u \leftarrow u + 1$
$K_u \leftarrow\!\!\$\ \mathcal{K}$
$\sigma_u \leftarrow 0$
$Y_u \leftarrow\!\!\$\ \{0,1\}^{|K_u|}$
if $(K_u, y) \in S$
  $\mathsf{bad} \leftarrow \mathsf{true}$
  $\boxed{Y_u \leftarrow y}$
$S \leftarrow S \cup \{(K_u, Y_u)\}$

$\underline{\text{Oracle } \textsc{Enc}(M, A, i)}$

if $i \notin [u]$ **then**
  return $\perp$
if $\sigma_i \leq \lceil |K_i|/(4+\delta) \rceil$
  $\mathrm{len} \leftarrow Y_u[(4+\delta)\sigma_i,$
    $(4+\delta)\sigma_i + 3]$
else
  $\mathrm{len} \leftarrow\!\!\$\ \{0, \ldots, 15\}$
$j \leftarrow 0$
do
  $M' \leftarrow \mathsf{pad}(M, \mathrm{len})$
  $(C, T) \leftarrow \mathcal{E}^+_{K_i}(M')$
  $X \leftarrow F_{K'_A}(T)$
  $b \leftarrow X = Y_u[(4+\delta)\sigma_i + 3,$
    $(4+\delta+1)\sigma_i - 1]$
  $j \leftarrow j + 1$
while (not b) and $(j < s)$
$\sigma_i \leftarrow \sigma_i + 1$
return (C, T)

Figure 5.1: Games $\mathrm{Sub}^{\mathrm{det}}_{\mathcal{E},\widetilde{\mathcal{E}}_{\delta,s}}(D)$, I_0, and I_1 used in the proof from Section 5.1.

$\boxed{I_2}$ I_3

$u \leftarrow 0$
$(K_A, K'_A) \leftarrow\!\!\$\ \widetilde{\mathcal{K}}$
$L \leftarrow \emptyset$
$b \leftarrow D^{\text{New},\text{Enc}}()$
return $b = 1$

Oracle New()

$u \leftarrow u + 1$
$\sigma_u \leftarrow 0$
$K_u \leftarrow\!\!\$\ \mathcal{K}$
$Y_u \leftarrow\!\!\$\ \{0,1\}^{|K_u|}$

Oracle Enc(M, A, i)

if $i \notin [u]$ **then**
  return $\perp$
if $\sigma_i \leq \lceil |K_i|/(4+\delta) \rceil$
  $\text{len} \leftarrow Y[(4+\delta)\sigma_i,$
    $(4+\delta)\sigma_i + 3]$
else
  $\text{len} \leftarrow\!\!\$\ \{0, \ldots, 15\}$
$j \leftarrow 0$
do
  $M' \leftarrow \mathsf{pad}(M, \text{len})$
  $(C, T) \leftarrow \mathcal{E}^+_{K_i}(M')$
  $X \leftarrow\!\!\$\ \{0,1\}^{\delta}$
  if $(T, x) \in L$
    $\mathsf{bad} \leftarrow \mathsf{true}$
    $\boxed{X \leftarrow x}$
  $L \leftarrow L \cup \{(T, X)\}$
  $b \leftarrow X = Y[(4+\delta)\sigma_i + 3,$
    $(4+\delta+1)\sigma_i - 1]$
  $j \leftarrow j + 1$
while (not b) and $(j < s)$
$\sigma_i \leftarrow \sigma_i + 1$
return (C, T)

$\text{Re}^{\text{det}}_{\mathcal{E},\widetilde{\mathcal{E}}_{\delta,s}}(D)$

$b \leftarrow D^{\text{New},\text{Enc}}()$
return $b = 1$

Oracle New()

$u \leftarrow u + 1$
$\sigma_u \leftarrow 0$
$K_u \leftarrow\!\!\$\ \mathcal{K}$

Oracle Enc(M, A, i)

if $i \notin [u]$ **then**
  return $\perp$
$(C, T) \leftarrow \mathcal{E}^+_{K_i}(M, \sigma_i)$
$\sigma_i \leftarrow \sigma_i + 1$
return (C, T)

Figure 5.2: Games I_2, I_3, and $\text{Re}^{\text{det}}_{\mathcal{E},\widetilde{\mathcal{E}}_{\delta,s}}(D)$ used in the proof from Section 5.1.

H_0 $\boxed{H_1}$

$(K, (K_A, K'_A)) \leftarrow\!\!\$\ \mathcal{K} \times \widetilde{\mathcal{K}}$
$\boxed{\Gamma \leftarrow\!\!\$\ \mathcal{F}(\{0,1\}^*, \{0,1\}^\delta)}$
$\sigma' \leftarrow \epsilon$
$r \leftarrow \lceil |K|/(4+\delta) \rceil$
$Y \leftarrow E(K_A, K)$
for $\sigma = 1, \ldots, r$
  $i \leftarrow 0$
  $b \leftarrow \mathsf{false}$
  $(M, \sigma') \leftarrow \mathcal{M}(\sigma')$
  $j \leftarrow 0$
  while (not b) and $(j < \delta)$
    $\text{len} \leftarrow\!\!\$\ \{0, \ldots, 15\}$
    $M' \leftarrow \mathsf{pad}(M, \text{len})$
    $(C, T) \leftarrow \mathcal{E}^+(K, M')$
    $X \leftarrow F(K'_A, T)$
    $\boxed{X \leftarrow \Gamma(T)}$
    $b \leftarrow X = Y[(4+\delta)\sigma + 3,$
      $(4+\delta)(\sigma+1) - 1]$
    $j \leftarrow j + 1$
  if not b
    return 1
return 0

$\boxed{H_2}$ H_3

$\mathsf{bad} \leftarrow \mathsf{false}$
$(K, K_A) \leftarrow\!\!\$\ \mathcal{K} \times \widetilde{\mathcal{K}}$
$\sigma' \leftarrow \epsilon$
$L \leftarrow \emptyset$
$r \leftarrow \lceil |K|/(4+\delta) \rceil$
$Y \leftarrow E(K_A, K)$
for $\sigma = 1, \ldots, r$
  $i \leftarrow 0$
  $b \leftarrow \mathsf{false}$
  $(M, \sigma') \leftarrow \mathcal{M}(\sigma')$
  $j \leftarrow 0$
  while (not b) and $(j < \delta)$
    $\text{len} \leftarrow\!\!\$\ \{0, \ldots, 15\}$
    $M' \leftarrow \mathsf{pad}(M, \text{len})$
    $(C, T) \leftarrow \mathcal{E}^+(K_E, M')$
    $X \leftarrow\!\!\$\ \{0,1\}^\delta$
    if $(T, x) \in L$
      $\mathsf{bad} \leftarrow \mathsf{true}$
      $\boxed{X \leftarrow x}$
    $L \leftarrow L \cup \{(T, X)\}$
    $b \leftarrow X = Y[(4+\delta)\sigma + 3,$
      $(4+\delta)(\sigma+1) - 1]$
    $j \leftarrow j + 1$
  if not b
    return 1
return 0

Figure 5.3: Games used in the proof from Section 5.2.

$$\leq q^2 s^2 2^{-\mathbf{H}_\infty(\mathcal{E}.\text{Auth})-1},$$

where $\mathcal{E}.\text{Auth}$ denotes the tag generation algorithm of $\mathcal{E}^+$. Finally, the Games I_3 and $\text{Re}^{\text{det}}_{\mathcal{E},\widetilde{\mathcal{E}}}(D)$ are identical. Indeed, in Game I_3, the condition that stops the while loop is completely independent from the generated ciphertext, which means that the distribution of the outputs of both encryption oracles are completely identical, and

$$\Pr(I_3) = \Pr\left(\text{Re}^{\text{det}}_{\mathcal{E},\widetilde{\mathcal{E}}}(D)\right).$$

5.2 Lower Bounding the Probability of Key Recovery

As the final part of the proof, our goal is to lower bound the success probability of $\widetilde{\mathcal{E}}_{s,\delta}$. In order to do so, we will use the sequence of games that are defined in Figure 5.3. First of all, we are going to switch our point of view and to consider the probability that it fails. Second, since the only possibility of failure comes from the new sampling mechanism for the content of the padding, we are simply going to replace the sampling of the padding

length by a uniformly random draw[6]. Now, we are going to start by replacing F in the pseudocode of $\widetilde{\mathcal{E}}$ by its ideal counterpart. Hence, one has

$$1 - \mathbf{Adv}^{\mathsf{kr}}_{\mathcal{M},q}(\widetilde{\mathcal{E}}_{s,\delta}) = \Pr(H_0) \leq \Pr(H_1) + \mathbf{Adv}^{\mathsf{prf}}_{F}(D'_F),$$

where D'_F is an adversary against the PRF-security of F that runs H_0, and replaces the calls to F by calls to its oracle. Hence, D'_F issues at most sq queries to its oracle, and runs in time $t + O(sql)$, where l is an upper bound on the number of bits that $\mathcal{M}(q)$ can output.

Game H_2 is identical to game H_1 since Γ has been replaced by its equivalent lazy sampling. Finally, game H_3 is identical to game H_2 until there exists a tag repetition, in which case game H_3 breaks the consistency with a simulated random function by sampling a new random value every time. By the fundamental Lemma of game playing [BR06], we have

$$\begin{aligned} 1 - \mathbf{Adv}^{\mathsf{kr}}_{\mathcal{M},q}(\widetilde{\mathcal{E}}_{s,\delta}) &= \Pr(H_1) + \mathbf{Adv}^{\mathsf{prf}}_{F}(D'_F) \\ &\leq \Pr(H_2) + \mathbf{Adv}^{\mathsf{prf}}_{F}(D'_F) \\ &\leq \Pr(H_3) + \Pr(H_3 \text{ sets } \mathsf{bad}) \\ &\quad + \mathbf{Adv}^{\mathsf{prf}}_{F}(D'_F). \end{aligned}$$

Clearly, the event H_3 sets bad implies that the game has created some tag collision in its at most qs encryption queries. Given that the inputs to $\mathcal{E}$ are nonce-respecting since $\mathcal{M}$ simulates a run of the MTProto protocol, one has

$$\Pr(H_3 \text{ sets } \mathsf{bad}) \leq q^2 s^2 2^{-\mathbf{H}_\infty(\mathcal{E}.\mathrm{Auth})-1},$$

where $\mathcal{E}.\mathrm{Auth}$ denotes the tag generation algorithm of $\mathcal{E}^+$.

The last remaining step is to upper bound the probability that H_3 returns 1. In that case, for every σ, the only way for H_3 to return 1 is if every draw of X fails to be equal to the corresponding bits of Y. Since a fresh uniformly random X is drawn at every execution of the loop, the probability that 1 is returned at each iteration of the for loop is thus

$$\left(1 - \frac{1}{2^\delta}\right)^s.$$

Overall, one has

$$\begin{aligned} 1 - \mathbf{Adv}^{\mathsf{kr}}_{\mathcal{M},q}(\widetilde{\mathcal{E}}_{s,\delta}) \leq q\left(1 - \frac{1}{2^\delta}\right)^s + q^2 s^2 2^{-\mathbf{H}_\infty(\mathcal{E}.\mathrm{Auth})-1} \\ + \mathbf{Adv}^{\mathsf{prf}}_{F}(D'_F). \end{aligned}$$

□

5.3 Impact of our Attack

The attack presented in the last section targets the MTProto AE scheme. We have to take into account the fact that this scheme is used in the broader MTProto protocol: a single key is used for the encryption of at most 100 messages. The dominating term from the success probability bound is clearly $q\left(1 - \frac{1}{2^\delta}\right)^s$, where $q \leq 100$. Table 5.1 presents several

[6] We stress that it holds because the part of the attack that exploits the randomized length of the padding cannot fail. We will not be able to use the same simplification when considering the strong undetectability of the attack.

possible choices of δ and s, and provides the associated probability to recover a number k of key bits given the number of victim queries.

Given the huge key size of the MTProto AE scheme and the numbers from Figure 5.1, targeting a full key recovery does not seem reasonable. Instead, if the goal of the adversary is to allow decryption of a high percentage of sent messages, one possible choice is to target the part of the MTProto key that is used during the encryption pass, which is 576 bits long. Assuming that the victim sends around 50 messages per key, choosing $\delta = 8$ and $s = 1485$ allows the complete recovery of the encryption key with a probability around 0.85. Of course, this comes at a computational cost: the subverted client will have to repeat the authentication step at most 1485 times. The increased energy consumption may become noticeable. A more modest choice ($\delta = 6$ and $s = 369$) will allow the recovery of most key bits with a high probability, while being computationally cheaper. Note that, even though the authentication pass has to be evaluated between 1 and s times, it is still possible to save the internal state of the SHA-256 hash function after the absorption of the message, and to start the authentication pass from this value for every choice of padding values. This reduces the computational overhead to its minimal value.

Table 5.1: This table presents an approximated lower bound on the probability to recover k bits of key material with the subversion attack $\widetilde{\mathcal{E}}_{\delta,s}$, under the assumption that the adversary does at least the specified number of queries.

δ	s	num. of queries	k	success probability
2	21	10	60	≥ 0.97
2	21	50	300	≥ 0.88
2	21	100	600	≥ 0.76
4	91	10	80	≥ 0.97
4	91	50	400	≥ 0.85
4	91	100	800	≥ 0.71
6	369	10	100	≥ 0.97
6	369	50	500	≥ 0.85
6	369	100	1000	≥ 0.70
8	1485	10	120	≥ 0.97
8	1485	50	600	≥ 0.85
8	1485	100	1200	≥ 0.70
10	5946	10	140	≥ 0.97
10	5946	50	700	≥ 0.85
10	5946	100	1400	≥ 0.70

Our attack requires access to a reliable counter. This is offered by the MTProto protocol in the case of secret chats. On the contrary, client-server chats do not offer such a convenient counter. There are possible workarounds for this issue:

- our algorithm can be made stateful, at the cost of making it detectable with a simple state reset;
- a randomized encryption scheme can be used instead of the length-preserving scheme E when computing Y in Algorithm 5.2; after each state reset, a new IV would be generated uniformly at random and stored as a state, along with a counter; doing this would reduce the number of sent key bits (both due to the need to send the IV bits, and to the fact that after resetting, the attacker will start sending the same key bits a second time), but it would also mitigate the impact of state resets;
- since the client-server key is long-lived, we can afford to transmit key bits very slowly; hence, Algorithm 2.2 can be used.

5.4 Instantiating F and E

Our subverted algorithm(s) require a length-preserving encryption scheme E and a PRF F. Here, we briefly discuss possible instantiation for these components.

Choice of F: From Table 5.1, we observe that efficient instances of our attacks set $\delta \leq 10$. So, we can simply reuse the AES-256 block cipher as F and truncate the output to δ bits. Since the output size δ is quite small, for all practical purposes, we can simply assume that truncated AES-256 is a perfect random function.

Choice of E: As for E, one can apply a wide-block block cipher, or a format-preserving encryption scheme to achieve the wPRF security requirement. However, these schemes are not flexible enough as they require the whole ciphertext (encrypted K) for correct decryption.

In fact, on a closer look, one can observe that all we need is a sufficiently long keystream to mask K. So, an efficient stream cipher, or block keystream generator like the Ctr mode are sufficient. Unfortunately, they require a seed, which makes them prone to detection via state reset. Instead, we instantiate E with an efficient *online encryption* scheme [BBKN01].

An online encryption scheme E' is a length-preserving encryption scheme that satisfies the online property: *for all key K_A, any input x is a prefix of another input x' if and only if $E'_{K_A}(x)$ is a prefix of $E'_{K_A}(x')$*. Essentially, the online property implies that we can encrypt and decrypt in an on-the-fly manner, i.e., as soon as a full block[7] of ciphertext is available, it can be decrypted based on the past ciphertext blocks. So, the online cipher could be an efficient trade-off between a format-preserving cipher and a stream cipher. Further note that, E' behaves close to uniform random function as long as its inputs do not share any prefixes. So, it is easy to see that $E'(K)$ will be uniform at random and independent across different sessions, as long as the first block of K do not collide across sessions.

Fortunately, one of the components in MTProto satisfies online property. IGE with some fixed IV value is a secure online cipher in the known plaintext setting. Now, within E, one can either instantiate IGE with AES-256 to reuse the MTProto components directly, or instantiate IGE with a smaller block cipher, say with 64-bit block, to reduce the amount of ciphertext required for correct decryption of any ciphertext bit.

Let $\widetilde{E}$ denote the block cipher used within E. Overall, one can show that E can be instantiated with IGE at the cost of $O\left(u^2/2^n\right) + \mathbf{Adv}^{\mathsf{prp}}_{\widetilde{E}}(u)$, where n denotes the block size of $\widetilde{E}$. A proof for this bound is easy to derive. Basically, the bound $O\left(u^2/2^n\right)$ comes from two cases: first, we get a $u^2/2^n$ term due to the probability that the first n bits of keys across two distinct sessions (or users) collide; and second, IGE behaves as a secure online cipher, in the known plaintext setting, up to a cost of $c^2u^2/2^n$, where $c = \lceil |K|/n \rceil$ is a small constant.

6 Averting Subversion of MTProto2.0

The most effective countermeasure is to get rid of the randomized padding algorithm. Simply changing the padding length randomization is not sufficient, as the value of the padding can always still be used to transmit δ bits of data per ciphertext (instead of $4 + \delta$ bits in Algorithm 5.2). Another possible countermeasure would be to derive the padding values in a deterministic way from the secret key (and possibly from the encrypted message), and then to verify its value as the last step of the decryption procedure. This would remove the ability of a subverted algorithm to manipulate the value and the length of the padding, and prevent our attack. In this section, we describe one such method and then go on to show that the modified algorithm is indeed subversion-resistant.

[7] Here, block is used in context of the underlying block cipher of E'.

It is well-known [AP19b, DFP15] that perfect decryptability is a necessary condition for any symmetric-key encryption scheme to be subversion-resistant. While perfect decryptability is a theoretical requirement, which is hard to maintain in practical settings, the attacks targeting decryption algorithms are in general practically inefficient. So, we assume perfect decryptability in our discussion. Additionally, Degabriele et al. also exhibited input-triggered subversion [DFP15]. Indeed, the adversary can also exploit the ambiguity in the message language to construct such attacks. Accordingly, it is necessary to have independence between the keys and the message distribution. We assume that the messages (including the protocol parameters) are sampled independently of the keys.

UNIQUE CIPHERTEXTS: Consider $(\mathcal{K}, \mathcal{A}, \mathcal{M}, \mathcal{T})$-deterministic authenticated encryption scheme $\mathcal{E}$. For any key K, any message M and any associated data A let $\mathcal{C}(K, M, A)$ be the set of all ciphertexts (C, T) pair such that $D(K, C, T, A)$ accepts with message M, meaning its output is M for some τ. We say that $\mathcal{E}$ has unique ciphertexts if the set $\mathcal{C}(K, M, A)$ is a singleton for all K, M, A, i.e., for each triple (K, A, M) there exists a unique ciphertext. In [BPR14], Bellare et al. showed that having unique ciphertext property is sufficient for averting subversion resistance in context of algorithm substitution attacks. So, all we need to do is to enable the property of unique ciphertexts for MTProto2.0.

6.1 Changes in MTProto2.0: MTProto-D

We reuse the notations from Section 4.1 and 4.2. We define a modified protocol called MTProto-D, by making four small changes in the definition of MTProto2.0:

1. First, we redefine $\mathsf{F}_{k_1}(x) := \mathsf{chop}_{\tau+4}(\mathsf{SHA\text{-}256}(k_1 \| x))$. Specifically, we extract an extra 4 bits at the tag generation stage which will be used later in padding algorithm.

2. Second, we redefine the protocol enriched message **random_bytes** $= 0^{128}$. This is done specifically to avoid adversary's control on the payload.

3. Third, we define $(t, \ell) = \mathsf{F}_{k_1}(\mathsf{X})$, where $|t| = \tau$ and $|\ell| = 4$.

4. Fourth, we change the padding algorithm by

   (a) first, redefining $\mathbf{Rand}(\mathsf{X}) := \ell$, where ℓ is viewed as a 4-bit integer value, i.e., $\ell \in \{0, \ldots, 15\}$. Since, F is a secure PRF, we can assume that for all practical purposes ℓ is uniform at random. So all we have done is eliminate the adversary's influence over the padding length, under the assumption of independence of message distribution.

   (b) second, sampling the **random_padding** in either one of the following way:

      i. Set **random_padding** $:= 10^{d-1}$, where d is the length of the padding; or

      ii. Set **random_padding** $:= F_{K'}(\mathsf{X})$ for some secure PRF F and a key K' independent of the other keys.

In terms of security, first we note that MTProto-D loses at most 4-bit security since we now release additional 4 bits via F. Second, it is worth noting that although the tag generation is done just over the protocol enriched message, i.e., X, this does not hamper the security of the protocol. This can be argued by the fact that we either set the random padding to a constant value, or it is fully dependent on X. In both cases the decryption of ciphertext blocks corresponding to the padding bits must either have a specific form or it must follow an exact deterministic relation with X, which acts as a verification step for the padding value. For the sake of simplicity we assume that padding follows 3.(b).i.

In Corollary 6.1, we show that MTProto-D is a subversion-resistant algorithm under the assumption of perfect decryptability and an independent message distribution.

Corollary 6.1. *Suppose the message distribution $\mathcal{M}$ is independent of keys and perfect decryptability holds. Then,* MTProto-D *is subversion-resistant in context of algorithm substitution attacks.*

Proof. First note that once we fix a message, the padding length is fixed (since MTProto-D is deterministic), whence the corresponding ciphertext length is fixed. Thus, for any triple (K, A, M), the set $\mathcal{C}(K, A, M)$ has ciphertexts of equal lengths. Further, for each message M, the ciphertext is generated over $M\|10^{d-1}$ for a fixed d (depends on just K and M). Thus, using the bijectivity property of IGE we conclude that $\mathcal{C}(K, M, A)$ is a singleton set and hence, using [BPR14, Theorem 4], MTProto-D is a subversion-resistant algorithm. □

6.2 A Note on the Independence of Message Distribution

In the proof of Corollary 6.1, we assume that the messages are independent of the keys. This is necessary to avoid input-triggered subversion [DFP15]. Basically, the adversary can influence the messages by introducing spurious space characters, rearrange certain letters in some word, or even reorder multiple lines in a message.[8]

However, we note that such attacks, although of theoretical interest, are not of great practical value. First, these attacks generally incur either a large latency or the number of key bits extracted is quite low. Second, and more importantly, on the decryption end one can employ language-specific techniques to detect any anomalies in the decrypted message. For instance, one can restrict the set of allowed characters, and look for unnecessary spaces before encryption itself. Also, based on the English language usage statistics, one can come up with a threshold value used to determine if the messages are tampered with or not. So in overly cautious applications, which should be the case for any secure messaging app, it is fairly plausible to assume that the messages are indeed independent of the keys.

References

[AMPS22] Martin R. Albrecht, Lenka Mareková, Kenneth G. Paterson, and Igors Stepanovs. Four Attacks and a Proof for Telegram. In *Security and Privacy – IEEE-S&P 2022, Proceedings*, pages 87–106, 2022.

[AMV15] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-Resilient Signature Schemes. In *Computer and Communications Security – ACM-CCS 2015, Proceedings*, pages 364–375, 2015.

[AP19a] Marcel Armour and Bertram Poettering. Substitution Attacks against Message Authentication. *IACR Trans. Symmetric Cryptol.*, 2019(3):152–168, 2019.

[AP19b] Marcel Armour and Bertram Poettering. Subverting Decryption in AEAD. In *Cryptography and Coding – IMACC 2019, Proceedings*, pages 22–41, 2019.

[BBKN01] Mihir Bellare, Alexandra Boldyreva, Lars R. Knudsen, and Chanathip Namprempre. Online Ciphers and the Hash-CBC Construction. In *Advances in Cryptology – CRYPTO 2001, Proceedings*, pages 292–309, 2001.

[BBT16] Mihir Bellare, Daniel J. Bernstein, and Stefano Tessaro. Hash-Function Based PRFs: AMAC and Its Multi-User Security. In *Advances in Cryptology – EUROCRYPT 2016, Proceedings, Part I*, pages 566–595, 2016.

[BDL+11] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-Speed High-Security Signatures. In *Cryptographic Hardware and Embedded Systems – CHES 2011, Proceedings*, pages 124–142, 2011.

[8] In context of Telegram this will correspond to reordering of chat messages.

[BJK15] Mihir Bellare, Joseph Jaeger, and Daniel Kane. Mass-Surveillance without the State: Strongly Undetectable Algorithm-Substitution Attacks. In *Computer and Communications Security – ACM-CCS 2015, Proceedings*, page 1431–1440, 2015.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In *Advances in Cryptology – CRYPTO 2016, Proceedings, Part II*, pages 123–153, 2016.

[BL16] Karthikeyan Bhargavan and Gaëtan Leurent. On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN. In *Computer and Communications Security – ACM-CCS 2016, Proceedings*, pages 456–467, 2016.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In *Advances in Cryptology – ASIACRYPT 2000, Proceedings*, pages 531–545, 2000.

[BPR14] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of Symmetric Encryption against Mass Surveillance. In *Advances in Cryptology – CRYPTO 2014, Proceedings*, pages 1–19, 2014.

[BR06] Mihir Bellare and Philip Rogaway. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In *Advances in Cryptology – EUROCRYPT 2006, Proceedings*, pages 409–426, 2006.

[BSKC19] Joonsang Baek, Willy Susilo, Jongkil Kim, and Yang-Wai Chow. Subversion in Practice: How to Efficiently Undermine Signatures. *IEEE Access*, 7:68799–68811, 2019.

[BWP+20] Sebastian Berndt, Jan Wichelmann, Claudius Pott, Tim-Henrik Traving, and Thomas Eisenbarth. ASAP: Algorithm Substitution Attacks on Cryptographic Protocols. *IACR Cryptol. ePrint Arch.*, 2020:1452, 2020.

[Cam78] C. M. Campbell. Design and Specifcation of Cryptographic Capabilities. In *Computer Security and Data Encryption Standard, National Bureau of Standards Special Publications 500-27*, pages 54–66, 1978.

[Cha21] Julia Chan. Top Apps Worldwide for January 2021 by Downloads. Sensor Tower Blog, 2021. Online: https://sensortower.com/blog/top-apps-worldwide-january-2021-by-downloads. Accessed: June 28, 2021.

[CHY20] Rongmao Chen, Xinyi Huang, and Moti Yung. Subvert KEM to Break DEM: Practical Algorithm-Substitution Attacks on Public-Key Encryption. In *Advances in Cryptology – ASIACRYPT 2020, Proceedings, Part II*, pages 98–128, 2020.

[CS14] Shan Chen and John P. Steinberger. Tight Security Bounds for Key-Alternating Ciphers. In *Advances in Cryptology – EUROCRYPT 2014, Proceedings*, pages 327–350, 2014.

[Dam89] Ivan Damgård. A Design Principle for Hash Functions. In *Advances in Cryptology – CRYPTO 1989, Proceedings*, pages 416–427, 1989.

[DFP15] Jean Paul Degabriele, Pooya Farshim, and Bertram Poettering. A More Cautious Approach to Security Against Mass Surveillance. In *Fast Software Encryption – FSE 2015, Revised Selected Papers*, pages 579–598, 2015.

[EMST76] William F. Ehrsam, Carl H. W. Meyer, John L. Smith, and Walter L. Tuchman. Message Verification and Transmission Error Detection by Block Chaining. Patent 4074066, USPTO, 1976.

[HS21] Philip Hodges and Douglas Stebila. Algorithm Substitution Attacks: State Reset Detection and Asymmetric Modifications. *IACR Trans. Symmetric Cryptol.*, 2021(2):389–422, 2021.

[JNP14] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In *Advances in Cryptology – ASIACRYPT 2014, Proceedings, Part II*, pages 274–288, 2014.

[JNPS18] Jérémy Jean, Ivica Nikolić, Thomas Peyrin, and Yannick Seurin. Deoxys 1.43, 2018.

[JNPS21] Jérémy Jean, Ivica Nikolić, Thomas Peyrin, and Yannick Seurin. The Deoxys AEAD Family. *J. Cryptol.*, 34(3):31, 2021.

[JO16] Jakob Jakobsen and Claudio Orlandi. On the CCA (in)Security of MTProto. In *Security and Privacy in Smartphones and Mobile Devices – SPSM@CCS 2016, Proceedings*, pages 113–116, 2016.

[LCWW18] Chi Liu, Rongmao Chen, Yi Wang, and Yongjun Wang. Asymmetric Subversion Attacks on Signature Schemes. In *Information Security and Privacy – ACISP 2018, Proceedings*, pages 376–395, 2018.

[Mer89] Ralph C. Merkle. One Way Hash Functions and DES. In *Advances in Cryptology – CRYPTO 1989, Proceedings*, pages 428–446, 1989.

[MN17] Bart Mennink and Samuel Neves. Optimal PRFs from Blockcipher Designs. *IACR Trans. Symmetric Cryptol.*, 2017(3):228–252, 2017.

[NIF21] Jack Nicas, Mike Isaac, and Sheera Frenkel. Millions Flock to Telegram and Signal as Fears Grow Over Big Tech. The New York Times, 2021. Online: https://www.nytimes.com/2021/01/13/technology/telegram-signal-apps-big-tech.html. Accessed: June 28, 2021.

[NIS15] NIST. Secure Hash Standard (SHS). Fedral Information Processing Standards Publication FIPS 180-4, National Institute of Standards and Technology, U. S. Department of Commerce, 2015.

[Pat91] Jacques Patarin. *Etude des Générateurs de Permutations Pseudo-aléatoires Basés sur le Schéma du DES*. PhD thesis, Université de Paris, 1991.

[Pat08] Jacques Patarin. The "Coefficients H" Technique. In *Selected Areas in Cryptography – SAC 2008, Revised Selected Papers*, pages 328–345, 2008.

[PGV93] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In *Advances in Cryptology - CRYPTO '93, Proceedings*, pages 368–378, 1993.

[RS06] Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In *Advances in Cryptology – EUROCRYPT 2006, Proceedings*, pages 373–390, 2006.

[RTYZ16] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Cliptography: Clipping the Power of Kleptographic Attacks. In *Advances in Cryptology – ASIACRYPT 2016, Proceedings, Part II*, pages 34–64, 2016.

[RTYZ17] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Generic Semantic Security against a Kleptographic Adversary. In *Computer and Communications Security – ACM-CCS 2017, Proceedings*, pages 907–922, 2017.

[RTYZ18] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Correcting Subverted Random Oracles. In *Advances in Cryptology – CRYPTO 2018, Proceedings, Part II*, pages 241–271, 2018.

[SBK+17] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The First Collision for Full SHA-1. In *Advances in Cryptology – CRYPTO 2017, Proceedings, Part I*, pages 570–596, 2017.

[Sim83] Gustavus J. Simmons. The Prisoners' Problem and the Subliminal Channel. In *Advances in Cryptology – CRYPTO 1983, Proceedings*, pages 51–67, 1983.

[The21a] The Telegram Team. FAQ for the Technically Inclined – What about IND-CCA? Telegram website, 2021. Online: https://core.telegram.org/techfaq/mtproto_v1#what-about-ind-cca. Accessed: June 28, 2021.

[The21b] The Telegram Team. FAQ for the Technically Inclined – Why did you use a custom protocol? Telegram website, 2021. Online: https://core.telegram.org/techfaq#q-why-did-you-go-for-a-custom-protocol. Accessed: June 28, 2021.

[The21c] The Telegram Team. Mobile Protocol. Telegram website, 2021. Online: https://core.telegram.org/mtproto. Accessed: June 28, 2021.

[The21d] The Telegram Team. Moving Chat History from Other Apps. Telegram News, 2021. Online: https://telegram.org/blog/move-history. Accessed: June 28, 2021.

[YY96] Adam L. Young and Moti Yung. The Dark Side of "Black-Box" Cryptography, or: Should We Trust Capstone? In *Advances in Cryptology – CRYPTO 1996, Proceedings*, pages 89–103, 1996.

[YY97] Adam L. Young and Moti Yung. Kleptography: Using Cryptography Against Cryptography. In *Advances in Cryptology – EUROCRYPT 1997, Proceeding*, pages 62–74, 1997.