How Secure is TextSecure?

Tilman Frosch Christian Mainka Christoph Bader
Florian Bergsma Jörg Schwenk Thorsten Holz
Horst Görtz Institute for IT Security, Ruhr University Bochum
{firstname.lastname}@hgi.rub.de

Abstract

Instant Messaging has attracted a lot of attention by users for both private and business communication and has especially gained popularity as low-cost short message replacement on mobile devices. However, most popular mobile messaging apps do not provide end-to-end security. Press releases about mass surveillance performed by intelligence services such as NSA and GCHQ lead many people looking for means that allow them to preserve the security and privacy of their communication on the Internet. Additionally fueled by Facebook's acquisition of the hugely popular messaging app WHATSAPP, alternatives that claim to provide secure communication experienced a significant increase of new users.

A messaging app that has attracted a lot of attention lately is TEXTSECURE, an app that claims to provide secure instant messaging and has a large number of installations via Google's Play Store. It's protocol is part of Android's most popular aftermarket firmware CYANOGENMOD. In this paper, we present the first complete description of TEXTSECURE's complex cryptographic protocol and are the first to provide a thorough security analysis of TEXTSECURE. Among other findings, we present an Unknown Key-Share Attack on the protocol, along with a mitigation strategy, which has been acknowledged by TEXTSECURE's developers. Furthermore, we formally prove that—if our mitigation is applied—TEXTSECURE's push messaging can indeed achieve the goals of authenticity and confidentiality.

I. INTRODUCTION

Since more than a decade, *Instant Messaging* (IM) has attracted a lot of attention by users for both private and business communication. IM has several advantages over classical e-mail communication, especially due to the chat-like user interfaces provided by popular tools. However, compared to the security mechanisms available for e-mail such as PGP [1] and s/MIME [2], text messages were sent unprotected in terms of authenticity and confidentiality on the Internet by the corresponding IM tools: in the early days, many popular IM solutions like MSN MESSENGER and YAHOO MESSENGER did not provide any security mechanisms at all. AOL only added a protection mechanism similar to s/MIME to their IM service later on and Trillian's SECUREIM messenger encrypted the data without providing any kind of authentication. Nowadays, most clients provide at least client-to-server encryption via TLS. Mechanisms like *Off the Record* (OTR) communication [3] are available that provide among other security properties end-to-end confidentiality.

As the popularity of smartphones grows, the Internet is accessible almost everywhere and mobile communication services gained a lot of attraction. IM is one of the most popular services for mobile devices and apps like WHATSAPP and SKYPE are among the top downloaded apps in the popular app stores. Unfortunately, both applications are closed-source and it is unknown which security mechanisms—beside a proprietary or TLS-based client-to-server encryption—are implemented. As such, it is hard to assess which kind of security properties are provided by these apps and especially end-to-end encryption is missing. In the light of the recent revelations of mass surveillance actions performed by intelligence services such as NSA and GCHQ, several secure IM solutions that are not prone to surveillance and offer a certain level of security were implemented.

One of the most popular apps for *secure* IM is TEXTSECURE, an app developed by *Open WhisperSystems* that claims to support end-to-end encryption of text messages. While previously focussing on encrypted short message service (SMS) communication, *Open WhisperSystems* introduced data channel-based push messaging in February 2014. Thus, the app offers both an iMessage- and WhatsApp-like communication mode, providing SMS+data channel or data channel-only communications [4]. Following Facebook's acquisition of WHATSAPP, TEXTSECURE gained a lot of popularity among the group of privacy-concious users and has currently more than 500,000 installations via *Google Play*. Its encrypted messaging protocol has also been integrated into the OS-level SMS-provider of CyanogenMod [5], a popular open source aftermarket Android firmware that has been installed on about 10 million Android devices [6]. Despite this popularity, the messaging protocol behind TEXTSECURE has not been rigorously reviewed so far. While the developers behind TEXTSECURE have a long history of research in computer security [7]–[12] and TEXTSECURE has received praise by whistleblower Edward Snowden [13], a security assessment is needed to carefully review the approach.

In this paper, we perform a thorough security analysis of TEXTSECURE's protocol. To this end, we first review the actual security protocol implemented in the app and provide a precise mathematical description of the included security primitives. Based on this protocol description, we perform a security analysis of the protocol and reveal an *Unknown Key-Share attack*, an attack vector first introduced by Diffie et. al. [14]. To the best of our knowledge, we are the first to discuss an actual attack against TEXTSECURE. We also reveal several other (minor) security problems in the current version of TEXTSECURE. Based on these findings, we propose a mitigation strategy that prevents the UKS attack. Furthermore, we also formally prove that TEXTSECURE with our mitigation strategy is secure and achieves *one-time stateful authenticated encryption*.

In summary, we make the following contributions:

- We are the first to completely and precisely document and analyze TEXTSECURE's secure push messaging protocol.
- We found an Unknown Key-Share attack against the protocol. We have documented the attack and show how it can be mitigated. The attack has been communicated with and and acknowledged by the developers of TEXTSECURE. We show that our proposed method of mitigation actually solves the issue.
- We show that if long-term public keys are authentic, so are the message keys, and that the encryption block of TEXTSECURE is actually one-time stateful authenticated encryption. Thus, we prove that TEXTSECURE's push messaging can indeed achieve the goals of authenticity and confidentiality.

II. TEXTSECURE PROTOCOL

We start with a precise description of the protocol implemented by TEXTSECURE. We obtained this information by analyzing the source code of the Android app and recovering the individual building blocks of the protocol. TEXTSECURE builds upon a set of cryptographic primitives. For ECDH operations, these are Curve25519 [15] as implemented in Google's Android Native Library. For symmetric encryption, TEXTSECURE relies on AES in both counter mode without padding and cipher block chaining mode with PKCS5 padding. For authenticity and integrity, HMACSHA256 is used. Security considerations of the cryptographic primitives are not within the scope of this paper.

For push messaging via data channel, TEXTSECURE relies on a central server[1] ($\mathcal{TS}$) to relay messages to the intended recipient. Parties communicate with $\mathcal{TS}$ via a REST-API using HTTPS. $\mathcal{TS}$'s certificate is self-signed, the certificate of the signing CA is hard-coded in the TEXTSECURE app. Actual message delivery is performed via Google Cloud Messaging (GCM), which basically acts as a router for messages.

A. TextSecure Protocol Flow

TEXTSECURE's protocol consists of several phases. We distinguish (i) *registration*, (ii) *sending/receiving a first message*, (iii) *sending a follow-up message*, and (iv) *sending a reply*.

Before a client is able to communicate, it needs to generate key material and register with $\mathcal{TS}$. When a party $\mathcal{P}_a$ first decides to use TEXTSECURE's data channel communication, it chooses an asymmetric long-term key pair (a, g^a), referred to as *identity key* by TEXTSECURE's developers. It also uses SHA1PRNG as provided by the Android Native Library to choose a password (pw), a registration ID (regID_a), and two keys $k_{enc,signal,a}$, $k_{mac,signal,a}$, each of 128 bit length. Additionally, the client chooses 100 asymmetric ephemeral key pairs, so-called *prekeys*, and one asymmetric *last resort key* (k_{lr}). When a party calculates a message authentication code (MAC), it uses HMACSHA256 as implemented by Android's respective Native Library.

[1] textsecure-service.whispersystems.org

For a party $\mathcal{P}_a$ to send a message to a party $\mathcal{P}_b$, $\mathcal{P}_a$ requests one of $\mathcal{P}_b$'s public prekeys from $\mathcal{TS}$, uses it to derive a shared secret, forms a message, whereof parts are encrypted and/or protected by a MAC, authenticates with $\mathcal{TS}$, and transmits the message to $\mathcal{TS}$. $\mathcal{TS}$ shares a symmetric long-term key $(k_{enc,signal,b}, k_{mac,signal,b})$ with $\mathcal{P}_b$, which it uses to encrypt all parts of $\mathcal{P}_a$'s message that are to be transmitted to $\mathcal{P}_b$. $\mathcal{TS}$ then hands off this encrypted message to GCM for delivery to $\mathcal{P}_b$. If $\mathcal{P}_a$ wants to send a follow-up message to $\mathcal{P}_b$, it derives a new key using a function f that is seeded with existing key material. When a party does not merely send a follow-up message, but a reply within a conversation, it also introduces new entropy into the seed of f and transmits a new ephemeral public key.

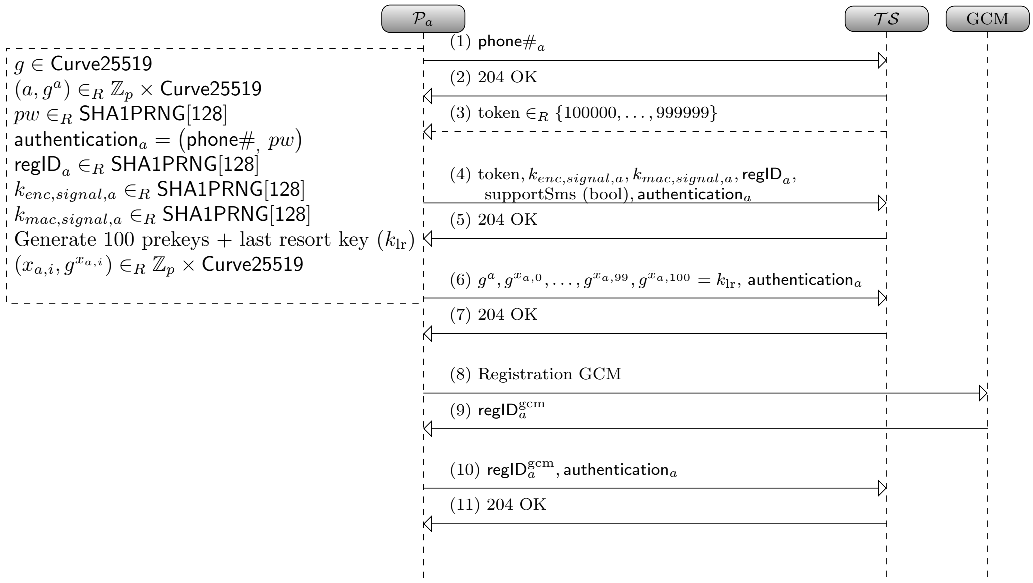

$\mathcal{P}_a$
$\mathcal{TS}$
GCM
$g \in$ Curve25519
$(a, g^a) \in_R \mathbb{Z}_p \times$ Curve25519
$pw \in_R$ SHA1PRNG[128]
authentication$_a = (\text{phone}\#, pw)$
regID$_a \in_R$ SHA1PRNG[128]
$k_{enc,signal,a} \in_R$ SHA1PRNG[128]
$k_{mac,signal,a} \in_R$ SHA1PRNG[128]
Generate 100 prekeys + last resort key (k_{lr})
$(x_{a,i}, g^{x_{a,i}}) \in_R \mathbb{Z}_p \times$ Curve25519
(1) phone$\#_a$
(2) 204 OK
(3) token $\in_R \{100000, \dots, 999999\}$
(4) token, $k_{enc,signal,a}$, $k_{mac,signal,a}$, regID$_a$, supportSms (bool), authentication$_a$
(5) 204 OK
(6) $g^a, g^{\bar{x}_{a,0}}, \dots, g^{\bar{x}_{a,99}}, g^{\bar{x}_{a,100}} = k_{\text{lr}}$, authentication$_a$
(7) 204 OK
(8) Registration GCM
(9) regID$_a^{\text{gcm}}$
(10) regID$_a^{\text{gcm}}$, authentication$_a$
(11) 204 OK

Figure 1: TEXTSECURE registration.

B. Detailed Description of Messages

In the following we give a detailed description of messages sent and processed in the different phases, as well as the key derivation.

1) Registration: The registration process is depicted in detail in Figure 1. To register with TEXTSECURE, a party $\mathcal{P}_a$ requests a verification token by transmitting its phone number (phone$\#_a$) and its preferred form of transport to $\mathcal{TS}$ (Step 1), which $\mathcal{TS}$ confirms with a HTTP status 204 (Step 2). Depending on the transport $\mathcal{P}_a$ chose, $\mathcal{TS}$ then dispatches either a short message or a voice call containing a random token (Step 3) to the number transmitted in Step 1. $\mathcal{P}_a$ performs the actual registration in Step 4, where it shows ownership of phone$\#_a$ by including the token, registers its credentials with the server via HTTP basic authentication [16], and sets its *signaling keys*. In this step, the client also states whether it wishes to communicate only via data channel push message or also accepts short messages. The server accepts if the token corresponds to the one supplied in Step 3 and the phone number has not been registered yet.

In Step 6, $\mathcal{P}_a$ supplies its 100 *prekeys* and k_{lr} to $\mathcal{TS}$. Prekeys are not transmitted individually, but within a JSON structure consisting of a *keyID* z, a *prekey* $g^{x_{a,i}}$, and the long-term key g^a. The *last resort key* is transmitted in the same way and identified by *keyID* 0xFFFFFF. The server accepts, if the message is well-formed and HTTP basic authentication is successful. $\mathcal{P}_a$ then registers with GCM (Step 8) and receives its regID$_a^{\text{gcm}}$ (Step 9), which it transmits to $\mathcal{TS}$ in Step 10 after authenticating again.

2) Sending an Initial Message: We define the period in which $\mathcal{P}_a$ employs one *prekey* to communicate with $\mathcal{P}_b$ as a *session*. When a new session is created to exchange messages, three main cryptographic building blocks are applied: a) a key exchange protocol with implicit authentication to exchange a secret, b) a key update and management protocol (the so-called axolotl ratchet [17]), which updates the encryption and MAC keys for every outgoing message, and c) a stateful authenticated encryption scheme. The process is depicted in detail in Figure 2.

Intuitively, the key exchange is a triple Diffie-Hellman (DH) key exchange using long-term and ephemeral secret keys. This is the only step in the protocol flow that uses the long-term keys.

According to the developers [17], the key management protocol provides both forward secrecy (which roughly means that *past* sessions remain secure even if the long-term key of a party is corrupted) and future secrecy (which translates to the idea that even after leakage of a currently used shared key *future* keys will remain secure).

The result of the key exchange and key management is input to the stateful authenticated encryption scheme [18]. The state of the encryption scheme is provided by the key management system and handed over from every call of the encryption and decryption algorithm, respectively, to the next for the whole session. Every new message is encrypted under a fresh key. The scheme guarantees confidentiality and authenticity of the exchanged messages, which we discuss in detail in Section IV.

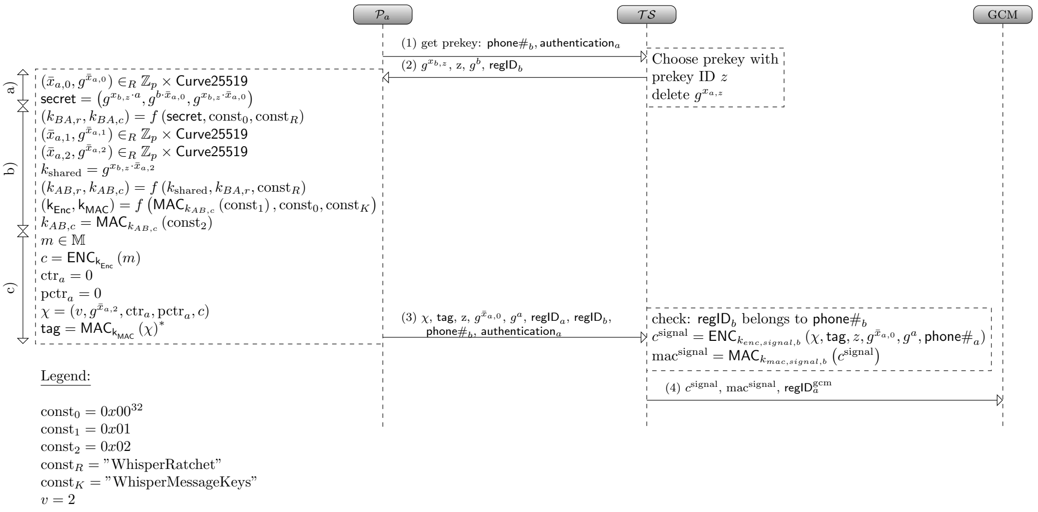

$\mathcal{P}_a$
$\mathcal{TS}$
GCM
(1) get prekey: $\mathsf{phone\#}_b, \mathsf{authentication}_a$
(2) $g^{x_{b,z}}$, z, g^b, regID_b
Choose prekey with prekey ID z delete $g^{x_{a,z}}$
a)
b)
c)
$(\bar{x}_{a,0}, g^{\bar{x}_{a,0}}) \in_R \mathbb{Z}_p \times \mathsf{Curve25519}$
$\mathsf{secret} = \left(g^{x_{b,z}\cdot a}, g^{b\cdot\bar{x}_{a,0}}, g^{x_{b,z}\cdot\bar{x}_{a,0}}\right)$
$(k_{BA,r}, k_{BA,c}) = f\,(\mathsf{secret}, \mathrm{const}_0, \mathrm{const}_R)$
$(\bar{x}_{a,1}, g^{\bar{x}_{a,1}}) \in_R \mathbb{Z}_p \times \mathsf{Curve25519}$
$(\bar{x}_{a,2}, g^{\bar{x}_{a,2}}) \in_R \mathbb{Z}_p \times \mathsf{Curve25519}$
$k_{\mathrm{shared}} = g^{x_{b,z}\cdot\bar{x}_{a,2}}$
$(k_{AB,r}, k_{AB,c}) = f\,(k_{\mathrm{shared}}, k_{BA,r}, \mathrm{const}_R)$
$(\mathsf{k_{Enc}}, \mathsf{k_{MAC}}) = f\left(\mathsf{MAC}_{k_{AB,c}}(\mathrm{const}_1), \mathrm{const}_0, \mathrm{const}_K\right)$
$k_{AB,c} = \mathsf{MAC}_{k_{AB,c}}(\mathrm{const}_2)$
$m \in \mathbb{M}$
$c = \mathsf{ENC}_{\mathsf{k_{Enc}}}(m)$
$\mathrm{ctr}_a = 0$
$\mathrm{pctr}_a = 0$
$\chi = (v, g^{\bar{x}_{a,2}}, \mathrm{ctr}_a, \mathrm{pctr}_a, c)$
$\mathsf{tag} = \mathsf{MAC}_{\mathsf{k_{MAC}}}(\chi)^*$
(3) χ, tag, z, $g^{\bar{x}_{a,0}}$, g^a, regID_a, regID_b, $\mathsf{phone\#}_b$, $\mathsf{authentication}_a$
check: regID_b belongs to $\mathsf{phone\#}_b$
$c^{\mathrm{signal}} = \mathsf{ENC}_{k_{enc,signal,b}}(\chi, \mathsf{tag}, z, g^{\bar{x}_{a,0}}, g^a, \mathsf{phone\#}_a)$
$\mathrm{mac}^{\mathrm{signal}} = \mathsf{MAC}_{k_{mac,signal,b}}\left(c^{\mathrm{signal}}\right)$
(4) c^{signal}, $\mathrm{mac}^{\mathrm{signal}}$, $\mathsf{regID}_a^{\mathrm{gcm}}$
Legend:
$\mathrm{const}_0 = 0x00^{32}$
$\mathrm{const}_1 = 0x01$
$\mathrm{const}_2 = 0x02$
$\mathrm{const}_R =$ "WhisperRatchet"
$\mathrm{const}_K =$ "WhisperMessageKeys"
$v = 2$

Figure 2: Sending an initial TEXTSECURE message.

a) Key Exchange: In the first step, $\mathcal{P}_a$ requests a *prekey* for $\mathcal{P}_b$ and receives a JSON structure consisting of prekeyID z, a prekey $g^{x_{b,z}}$, and $\mathcal{P}_b$'s long-term key g^b. $\mathcal{P}_a$ also receives regID_b from $\mathcal{TS}$ an then chooses a new ephemeral key to calculate a secret as the concatenation of three DH operations, combining $\mathcal{P}_b$'s prekey, $\mathcal{P}_a$'s long-term key, $\mathcal{P}_b$'s long-term key, and $\mathcal{P}_a$'s freshly chosen ephermeral key.

b) Key Management (axolotl ratchet): After $\mathcal{P}_a$ has completed the initial key exchange, it derives two symmetric keys $(k_{BA,r}, k_{BA,c})$ for receiving messages using f (cf. Algorithm 1). f is here seeded with secret. For all respective parameters of f see Figure 2. $\mathcal{P}_a$ then chooses a new ephemeral keypair $(\bar{x}_{a,1}, g^{\bar{x}_{a,1}})$, which is never used and just exists because of reuse. It then chooses another ephemeral keypair $(\bar{x}_{a,2}, g^{\bar{x}_{a,2}})$, which it uses to calculate k_{shared} as the output of a DH operation that takes $\mathcal{P}_b$'s prekey $g^{x_{b,z}}$ and $\bar{x}_{a,2}$ as input. $\mathcal{P}_a$ then derives two symmetric keys $(k_{AB,r}, k_{AB,c})$ for sending messages. Here f is seeded with k_{shared} and $k_{BA,r}$. Finally, $\mathcal{P}_a$ uses f, seeded with $k_{AB,c}$, to derive the message keys $(\mathsf{k_{Enc}}, \mathsf{k_{MAC}})$ and in the end derives a new $k_{AB,c}$ as $\mathsf{MAC}_{k_{AB,c}}(\mathsf{const}_2)$, where $\mathsf{const}_2 = 0x02$.

Algorithm 1 $f\,(input, key, string)$

```
$k_{pr} \leftarrow \mathsf{MAC}_{key}\,(input)$
$k_0 \leftarrow \mathsf{MAC}_{k_{pr}}\,(string, 0x00)$
$k_1 \leftarrow \mathsf{MAC}_{k_{pr}}\,(k_0, string, 0x01)$
**return** $(k_0, k_1)$
```

c) Stateful Authenticated Encryption: A message $m \in \mathbb{M}$ is encrypted using AES in counter mode without padding as $c = \mathsf{ENC}_{\mathsf{k_{Enc}}}(m)$. $\mathcal{P}_a$ then forms message (3.) and thus calculates $\mathsf{tag} = \mathsf{MAC}_{\mathsf{k_{MAC}}}(\chi)$, where $\chi = (v, g^{\bar{x}_{a,2}}, \mathrm{ctr}_a, \mathrm{pctr}_a, c)$. v represents the protocol version and is set to $0x02$. For ordering messages within a conversation ctr and pctr are used. Both are initially set to 0. ctr is incremented with every message a party sends, while pctr is set to the value ctr carried in the message a party is replying to.

Upon receiving message (3.), $\mathcal{TS}$ checks if regID_b corresponds to $\mathsf{phone}\#_b$. It then encrypts the parts of message (3.) intended for $\mathcal{P}_b$ with $\mathcal{P}_b$'s signaling key, using AES in CBC mode with PKCS5 padding. $\mathcal{TS}$ additionally calculates a MAC over the result, which we denote as $\mathrm{mac}^{\mathrm{signal}}$. $\mathcal{TS}$ sends both, encrypted message data c^{signal} and $\mathrm{mac}^{\mathrm{signal}}$, to the GCM server, together with $\mathsf{regID}_b^{\mathrm{gcm}}$ as the receipient. The result of this additional encryption layer is that Google's Cloud Messaging servers will only be able to see the receipient but not the sender of the message.

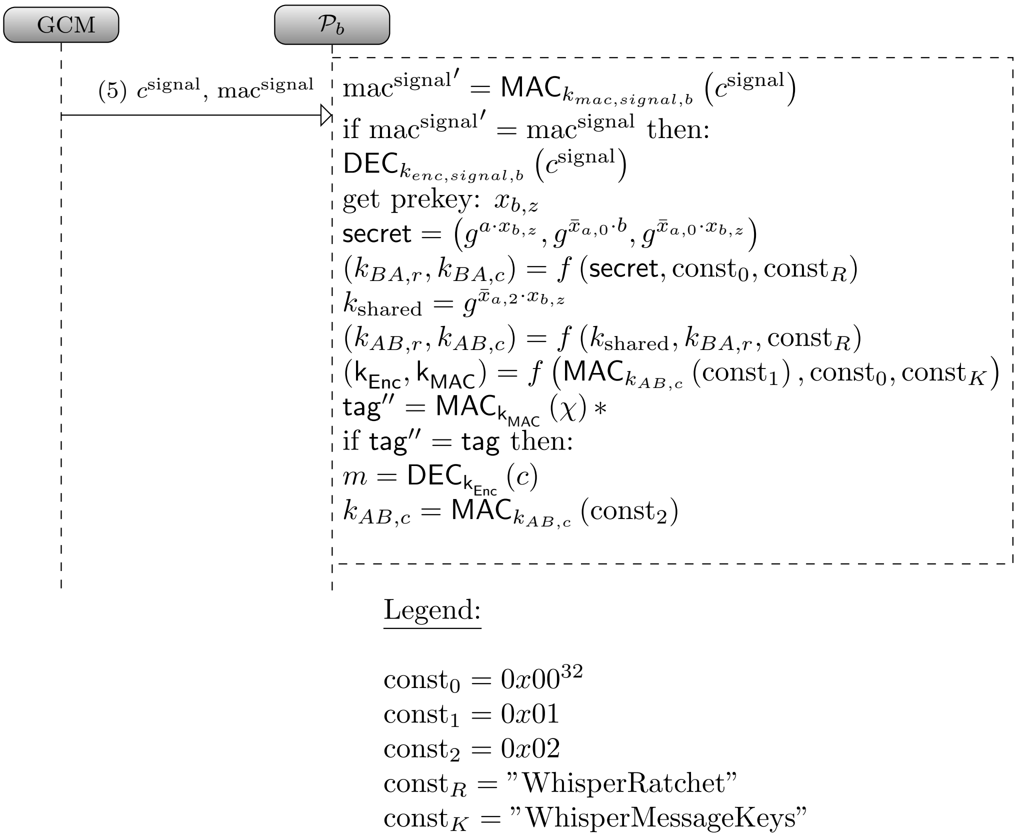

GCM
$\mathcal{P}_b$
(5) c^{signal}, $\mathrm{mac}^{\mathrm{signal}}$
$\mathrm{mac}^{\mathrm{signal}\prime} = \mathsf{MAC}_{k_{mac,signal,b}}\left(c^{\mathrm{signal}}\right)$
if $\mathrm{mac}^{\mathrm{signal}\prime} = \mathrm{mac}^{\mathrm{signal}}$ then:
$\mathsf{DEC}_{k_{enc,signal,b}}\left(c^{\mathrm{signal}}\right)$
get prekey: $x_{b,z}$
$\mathsf{secret} = \left(g^{a\cdot x_{b,z}}, g^{\bar{x}_{a,0}\cdot b}, g^{\bar{x}_{a,0}\cdot x_{b,z}}\right)$
$(k_{BA,r}, k_{BA,c}) = f\,(\mathsf{secret}, \mathrm{const}_0, \mathrm{const}_R)$
$k_{\mathsf{shared}} = g^{\bar{x}_{a,2}\cdot x_{b,z}}$
$(k_{AB,r}, k_{AB,c}) = f\,(k_{\mathrm{shared}}, k_{BA,r}, \mathrm{const}_R)$
$(\mathsf{k}_{\mathsf{Enc}}, \mathsf{k}_{\mathsf{MAC}}) = f\left(\mathsf{MAC}_{k_{AB,c}}(\mathrm{const}_1), \mathrm{const}_0, \mathrm{const}_K\right)$
$\mathsf{tag}'' = \mathsf{MAC}_{\mathsf{k}_{\mathsf{MAC}}}(\chi) *$
if $\mathsf{tag}'' = \mathsf{tag}$ then:
$m = \mathsf{DEC}_{\mathsf{k}_{\mathsf{Enc}}}(c)$
$k_{AB,c} = \mathsf{MAC}_{k_{AB,c}}(\mathrm{const}_2)$
Legend:
$\mathrm{const}_0 = 0x00^{32}$
$\mathrm{const}_1 = 0x01$
$\mathrm{const}_2 = 0x02$
$\mathrm{const}_R =$ "WhisperRatchet"
$\mathrm{const}_K =$ "WhisperMessageKeys"

Figure 3: Receiving an initial TEXTSECURE message.

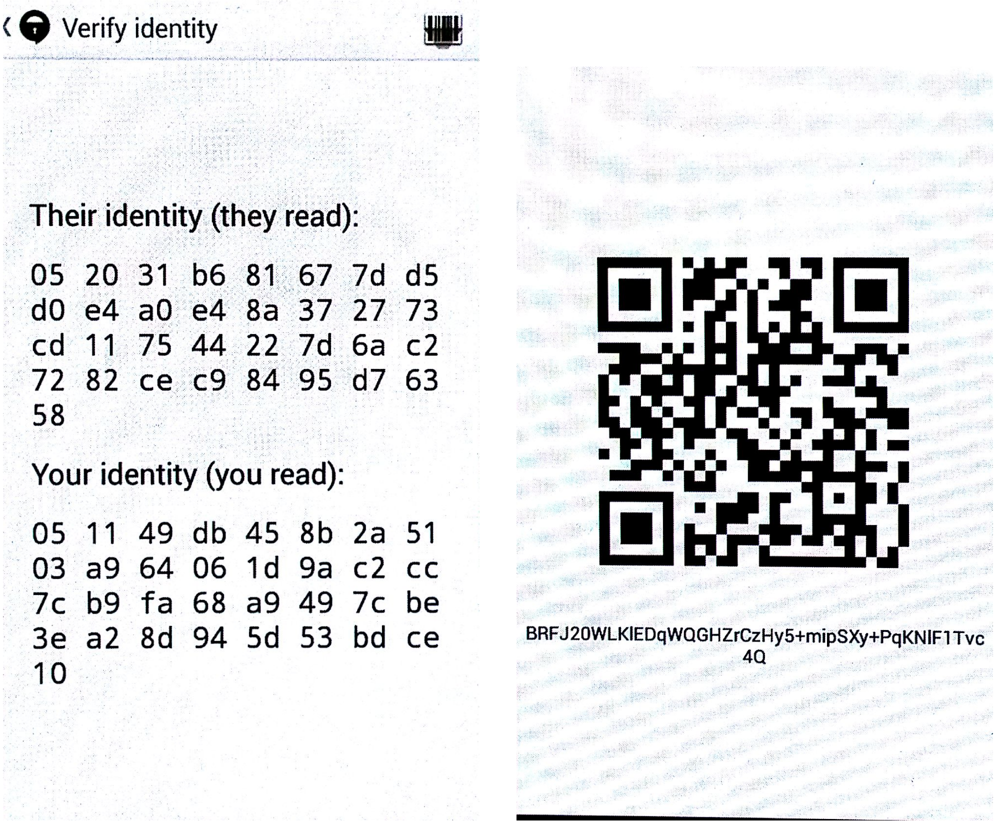
The receiving process is depicted in Figure 3. $\mathcal{P}_b$ receives the message in Step (5.). First, $\mathcal{P}_b$ verifies $\mathrm{mac}^{\mathrm{signal}}$ and, if successful, decrypts c^{signal}. It looks up its private key that corresponds to prekeyID z and calculates secret.

$\mathcal{P}_b$ then derives two symmetric keys $(k_{BA,r}, k_{BA,c})$ for sending messages by seeding f with secret. Afterwards, $\mathcal{P}_b$ calculates k_{shared} as the output of a DH operation that takes $\mathcal{P}_a$'s latest ephemeral key $g^{\bar{x}_{a,2}}$ and $\mathcal{P}_b$'s private prekey $x_{b,z}$ as input. In the next step $\mathcal{P}_b$ uses $f\,(k_{\mathsf{shared}}, k_{AB,r}, \mathsf{const}_R)$ to derive two symmetric keys $(k_{AB,r}, k_{AB,c})$ for receiving messages and derives the message keys $(\mathsf{k}_{\mathsf{Enc}}, \mathsf{k}_{\mathsf{MAC}})$.

$\mathcal{P}_b$ now verifies the MAC and, if successful, decrypts the message. In the end, $\mathcal{P}_b$ also derives a new $k_{AB,c} = \mathsf{MAC}_{k_{AB,c}}(\mathsf{const}_2)$.

3) Follow-up Message: If $\mathcal{P}_a$ follows up with a message before $\mathcal{P}_b$ replies, $\mathcal{P}_a$ derives a new pair $(\mathsf{k}_{\mathsf{Enc}}, \mathsf{k}_{\mathsf{MAC}}) = f\left(\mathsf{MAC}_{k_{AB,c}}(\mathsf{const}_1), \mathsf{const}_0, \mathsf{const}_K\right)$, which it then uses as detailed above.

4) Reply Message: If $\mathcal{P}_b$ wants to reply to a message within an existing session with $\mathcal{P}_a$, it first chooses a new ephemeral keypair $(\bar{x}_{b,0}, g^{\bar{x}_{b,0}})$ and calculates k_{shared} as the output of a DH operation that takes $\mathcal{P}_a$'s latest ephemeral public key $g^{\bar{x}_{a,2}}$ and its own freshly chosen ephemeral private key $\bar{x}_{b,0}$ as input. $\mathcal{P}_b$ then derives $(k_{BA,r}, k_{BA,c})$ by seeding f with k_{shared} and $k_{AB,r}$.



Verify identity

Their identity (they read):

```
05 20 31 b6 81 67 7d d5
d0 e4 a0 e4 8a 37 27 73
cd 11 75 44 22 7d 6a c2
72 82 ce c9 84 95 d7 63
58
```

Your identity (you read):

```
05 11 49 db 45 8b 2a 51
03 a9 64 06 1d 9a c2 cc
7c b9 fa 68 a9 49 7c be
3e a2 8d 94 5d 53 bd ce
10
```

```
BRFJ20WLKIEDqWQGHZrCzHy5+mipSXy+PqKNIF1Tvc
4Q
```

(a) Conversation fingerprints

(b) Own fingerprint QR code

Figure 4: TEXTSECURE fingerprint verification.

C. Key Comparison

In an attempt to establish that a given public key indeed belongs to a certain party, TEXTSECURE offers the possibility to display the fingerprint of a user's long-term public key. Two parties can then compare fingerprints using an out-of-band channel, for example, a phone call or an in-person meeting. If two parties meet in person, TEXTSECURE also offers to conveniently render the fingerprint of one's own long-term public key as a QR code, using a third-party application on Android, which the other party can then scan using the same application on their mobile device. TEXTSECURE then compares the fingerprint it just received to the party's fingerprint it received as part of a conversation. Figure 4 pictures these fingerprints.

III. ISSUES WITH TEXTSECURE

Based on the recovered protocol description, we can analyze its security properties. In the following, we discuss our findings.

A. MAC Image Space Only Partially Used

In Section II, we stated that TEXTSECURE uses HMACSHA256 to calculate MACs. Surprisingly, TEXTSECURE does not transmit the complete output of HMACSHA256. The message tag in Figures 2 and 3 does only represent the first 64 bit of the 256 bit MAC. Upon request, TEXTSECURE's developers stated that this just happens to reduce message size. However according to NIST [19, chapter 5.5], a MAC length of 80 bit is recommended when a MAC is used for *key confirmation*, which is the case in TEXTSECURE (at least) in the first message of a new session.

B. Unknown Key-Share Attack

An *Unknown Key-Share Attack* (UKS) is an attack vector first described by Diffie et. al. [14]. Informally speaking, if such an attack is mounted against $\mathcal{P}_a$, then $\mathcal{P}_a$ believes to share a key with $\mathcal{P}_b$, whereas in fact $\mathcal{P}_a$ shares a key with $\mathcal{P}_e \neq \mathcal{P}_b$.

For a better understanding how this can be related to TEXTSECURE, suppose the following example: Bart ($\mathcal{P}_b$) wants to trick his friend Milhouse ($\mathcal{P}_a$). Bart knows that Milhouse will invite him to his birthday party using TEXTSECURE (e.g., because Lisa already told him). He starts the UKS attack by replacing his own public key with Nelsons ($\mathcal{P}_e$) public key and lets Milhouse verify the fingerprint of his new public key. This can be justified, for instance, by claiming to have a new device and having simply re-registered, as that requires less effort than restoring an encrypted backup of the existing key material. Now, as explained in more detail below, if Milhouse invites Bart to his birthday party, then Bart may just forward this message to Nelson who will believe that this message was actually sent from Milhouse. Thus, Milhouse ($\mathcal{P}_a$) believes that he invited Bart ($\mathcal{P}_b$) to his birthday party, where in fact, he invited Nelson ($\mathcal{P}_e$).

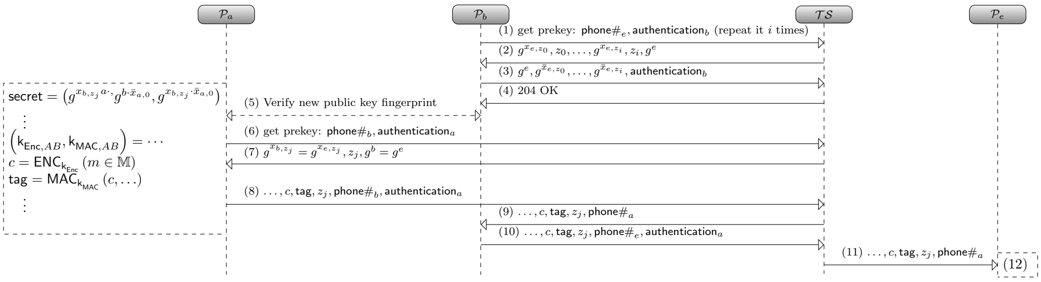

$\mathcal{P}_a$
$\mathcal{P}_b$
$\mathcal{TS}$
$\mathcal{P}_e$
(1) get prekey: $\mathsf{phone\#}_e, \mathsf{authentication}_b$ (repeat it i times)
(2) $g^{x_{e,z_0}}, z_0, \ldots, g^{x_{e,z_i}}, z_i, g^e$
(3) $g^e, g^{\bar{x}_{e,z_0}}, \ldots, g^{\bar{x}_{e,z_i}}, \mathsf{authentication}_b$
(4) 204 OK
(5) Verify new public key fingerprint
(6) get prekey: $\mathsf{phone\#}_b, \mathsf{authentication}_a$
(7) $g^{x_{b,z_j}} = g^{x_{e,z_j}}, z_j, g^b = g^e$
(8) $\ldots, c, \mathsf{tag}, z_j, \mathsf{phone\#}_b, \mathsf{authentication}_a$
(9) $\ldots, c, \mathsf{tag}, z_j, \mathsf{phone\#}_a$
(10) $\ldots, c, \mathsf{tag}, z_j, \mathsf{phone\#}_e, \mathsf{authentication}_a$
(11) $\ldots, c, \mathsf{tag}, z_j, \mathsf{phone\#}_a$
(12)
$\mathsf{secret} = (g^{x_{b,z_j}a \cdot}, g^{b \cdot \bar{x}_{a,0}}, g^{x_{b,z_j} \cdot \bar{x}_{a,0}})$
$\vdots$
$\left(\mathsf{k}_{\mathsf{Enc},AB}, \mathsf{k}_{\mathsf{MAC},AB}\right) = \cdots$
$c = \mathsf{ENC}_{\mathsf{k}_{\mathsf{Enc}}}(m \in \mathbb{M})$
$\mathsf{tag} = \mathsf{MAC}_{\mathsf{k}_{\mathsf{MAC}}}(c, \ldots)$
$\vdots$

Figure 5: UKS attack on TEXTSECURE: $\mathcal{P}_a$ believes to share a key with $\mathcal{P}_b$ but shares one with $\mathcal{P}_e$.

In detail, the attacker (Bart, $\mathcal{P}_b$) has to perform the steps shown in Figure 5 for this attack (only the important protocol parameters and steps are mentioned):

(1-2) $\mathcal{P}_b$ requests $g^{x_{e,z_0}}, \ldots, g^{x_{e,z_i}}$ from $\mathcal{TS}$ using $\mathsf{phone\#}_e$.

(3-4) $\mathcal{P}_b$ commits $g^{x_{e,z_0}}, \ldots, g^{x_{e,z_i}}$ to $\mathcal{TS}$ as his own prekeys plus g^e as its own long-term public key.

(5) $\mathcal{P}_b$ lets $\mathcal{P}_a$ verify the fingerprint of its new public key g^e. Note that this step uses QR-codes is is thus offline.

(6-7) Once $\mathcal{P}_a$ wants to send the message to $\mathcal{P}_b$, $\mathcal{P}_a$ requests a prekey for $\mathcal{P}_b$ by using $\mathsf{phone\#}_b$. $\mathcal{TS}$ returns $g^{x_{b,z_j}} = g^{x_{e,z_j}}$ and the long-term key $g^b = g^e$.

(8-9) $\mathcal{P}_a$ computes the secret using $g^{x_{b,z_j}}$ and g^b from which $\left(\mathsf{k}_{\mathsf{Enc},AB}, \mathsf{k}_{\mathsf{MAC},AB}\right)$ are going to be derived. For computing those keys, he uses in fact $\mathcal{P}_e$'s prekey and idenity key although he believes to use $\mathcal{P}_b$'s ones. He then encrypts message $m \in \mathbb{M}$, computes the respective MAC tag, and sends it to $\mathcal{P}_b$ (GCM ommited).

(10-11) $\mathcal{P}_b$ is neither able to verify the tag nor to decrypt the message c. He sends the ciphertext and message tag to $\mathcal{P}_e$.

(12) $\mathcal{P}_e$ processes the incoming message as usual. He computes the same secret as $\mathcal{P}_a$, because $g^{x_{b,z_j}} = g^{x_{e,z_j}}$ and $g^b = g^e$. The secret is then used to compute $\left(\mathsf{k}_{\mathsf{Enc},AE} = \mathsf{k}_{\mathsf{Enc},AB}, \mathsf{k}_{\mathsf{MAC},AE} = \mathsf{k}_{\mathsf{MAC},AB}\right)$ so that $\mathcal{P}_e$ is able to read and verify the message.

In Step 10, $\mathcal{P}_b$ has to forward the message to $\mathcal{P}_e$, such that it appears to be sent by $\mathcal{P}_a$. Therefore, he needs to include $\mathsf{authentication}_a$ for $\mathcal{TS}$ to include $\mathsf{phone\#}_a$ in Step 11, so that $\mathcal{P}_e$ will receive $\mathsf{phone\#}_a$ with the forwarded message. This can be achieved in several ways:

- $\mathcal{TS}$ is corrupted. In this case, it is a trivial task to get or circumvent $\mathsf{authentication}_a$.
- If $\mathcal{TS}$ is benign, an attacker might be able to eavesdrop $\mathsf{authentication}_a$. Although TLS is used for all connections between clients and server, future or existing issues with TLS implementations [20]–[24] can not be ruled out and would allow for a compromise of $\mathsf{authentication}_a$. Another possibility to obtain $\mathsf{authentication}_a$ could be a governmental agency (legally) enforcing access to the TLS keys.
- In contrast to a party's other key material, the password is stored unencrypted and is not protected by TEXTSECURE's master password. Thus, the easiest possibility to realize

this attack might be for an attacker to recover the password for $\mathsf{authentication}_a$ from TEXTSECURE's preferences[2].

Devices are left unattended on tables in bars and clubs, users can be compelled to hand devices over in a traffic control, stop-and-frisk operations, immigration and customs, or when passing through airport security. The widely used Android Unlock Pattern has been shown to be a weaker protection than a three-digit PIN [25] and can often also easily be recovered from the smudges a user leaves on the screen during the unlock process [26].

C. Unknown Key-Share Attack Variant

In this variant, there is no need for the attacker to know $\mathsf{authentication}_a$. Instead, he must only be able to stop/intercept one message, for example, by controlling one active network element in the path between $\mathcal{P}_a$ and $\mathcal{TS}$, like a WiFi accesspoint.

The attack from the previous section makes $\mathcal{P}_a$ (the sender of a message) believe he shares a key with $\mathcal{P}_b$ (intended receiver of that message) while he in fact shares a key with $\mathcal{P}_e$ (actual receiver of the message). To this end, $\mathcal{P}_b$ had to replace his own public key by the public key of the intended receiver.

An attack on $\mathcal{P}_e$ is also possible, using similar techniques: Suppose the attacker replaces his own public key with the public key of the sender. Then any message that is sent from $\mathcal{P}_a$ to $\mathcal{P}_e$ may also originate from $\mathcal{P}_b$. This makes $\mathcal{P}_e$ (the receiver) believe in that he shares a key with $\mathcal{P}_b$ (claimed sender) while he actually shares a key with $\mathcal{P}_a$ (actual sender).

This is a practical issue in competitions where, for instance, the first to send the solution to $\mathcal{P}_e$ wins a prize.

To mount such attack using TEXTSECURE, $\mathcal{P}_b$ replaces his own public key with the public key of $\mathcal{P}_a$ and lets $\mathcal{P}_e$ verify it. We stress again that replacing the public key and letting $\mathcal{P}_e$ verify it is not an issue for $\mathcal{P}_b$ in practice. Now, when $\mathcal{P}_a$ starts a new session with $\mathcal{P}_e$, $\mathcal{P}_b$ can mount the attack by intercepting the message sent from $\mathcal{P}_a$ to $\mathcal{P}_e$. He relays the message to $\mathcal{P}_e$, but uses his own $\mathsf{authentication}_b$.

Remark 1. *The problem of UKS attacks is not limited to* TEXTSECURE*, but a general issue in trust-models that do not rely on certification authorities. However, especially given that the pattern of UKS attacks is well-known in the literature [14], [27]–[30], we would have expected the* protocol *to thwart these attacks and to provide authenticity. This is not the case, as we have demonstrated above. We stress that UKS-attacks are always possible if two parties deliberately collaborate and, for instance, choose the same public key. However, the above attacks on* TEXTSECURE *work even if no two parties collaborate.*

D. Mitigation of Unknown Key-Share Attack

Let us consider the message that is sent in Step 8 of Figure 5:

$$\chi, \mathsf{tag}, g^{\bar{x}_{a,0}}, g^{a}, \mathsf{regID}_a, \mathsf{regID}_e,$$
$$\mathsf{phone\#}_e, \mathsf{authentication}_a,$$

where $\chi = (v, g^{\bar{x}_{a,2}}, \mathrm{ctr}_a, \mathrm{pctr}_a, c)$ and $\mathsf{tag} = \mathsf{MAC}_{\mathsf{k}_{\mathsf{MAC}}}(\chi)$. Intuitively, if both $\mathcal{P}_a$'s and $\mathcal{P}_e$'s identity were protected by the tag, then the attacks above do not longer work. As identities we propose to use the respective parties' phone numbers, as they represent a unique identifier within the system. χ would thus be formed as

$$(v, g^{\bar{x}_{a,2}}, \mathrm{ctr}_a, \mathrm{pctr}_a,$$
$$\mathsf{phone\#}_a, \mathsf{phone\#}_e, c).$$

If $\mathsf{k}_{\mathsf{MAC}}$ is secret (i.e., only shared among $\mathcal{P}_a$ and $\mathcal{P}_e$) and if MAC is secure, the inclusion of both identities in the tag provides a proof of $\mathcal{P}_a$ towards $\mathcal{P}_e$ that $\mathcal{P}_a$ is aware of $\mathcal{P}_e$ as its peer, i.e., that the message is indeed intended for $\mathcal{P}_e$. Moreover, $\mathcal{P}_e$ is convinced that $\mathcal{P}_a$ actually sent the message. Thus, $\mathcal{P}_b$ will not be able to mount the above attacks.

Remark 2. *Our mitigation resembles the concept of* strong entity authentication *[29]. However, this concept is not directly applicable here, since we consider asynchronous message exchange.*

[2] File: *shared-prefs/org.thoughtcrime.securesms_preferences.xml*

E. No Cryptographic Authentication

While the *Unknown Key-Share Attacks* are mitigated if the message in Step 8 is modified as we propose in Section III-D, the underlying problem is not resolved. It results from a party's erroneous assumption that a communication partner's long-term identity key is authentic, if they have compared key fingerprints and these fingerprints matched their assumptions. However, this is not necessarily the case. Given the attack scenario in Section III-B, a malicious party would always be able to present a third party's long-term public key as their own, as only fingerprints are compared – a party is not required to show their knowledge of the corresponding secret key.

F. Mitigation of Authentication Issue

In the following, we present two means of authentication. The first one provides a mutual cryptographic authentication with the help of digital signatures, while the second one achieves this goal requiring less effort in terms of implementation and also integrates seamlessly into TEXTSECURE's existing user experience.

1) Signature-based Cryptographic Authentication: $\mathcal{P}_a$ and $\mathcal{P}_b$ can establish the authenticity of their respective long-term keys as follows

1) $\mathcal{P}_a$ choses a token $r_A \in_R \mathsf{SHA1PRNG}[128]$ and presents a QR code containing r_A.
2) $\mathcal{P}_b$ scans this QR code, creates a signature σ over r_A using his long-term private key, chooses a token $r_B \in_R \mathsf{SHA1PRNG}[128]$ and creates a QR code containing both the signature over r_A and his own token.
3) $\mathcal{P}_a$ scans the QR code presented by $\mathcal{P}_b$ and verifies σ with respect to r_A using $\mathcal{P}_b$'s long-term public key. $\mathcal{P}_a$ then creates a signature σ' over (r_B, r_A) using his long-term private key and creates a QR code containing σ'.
4) $\mathcal{P}_b$ scans the QR code presented by $\mathcal{P}_a$ and verifies σ' with respect to (r_B, r_A) using $\mathcal{P}_a$'s long-term public key.

If the verification in Step 3 is successful, $\mathcal{P}_a$ is assured that $\mathcal{P}_b$'s long-term public key is authentic and $\mathcal{P}_b$ does know the corresponding private key. Likewise, if the verification in Step 4 is successful, $\mathcal{P}_b$ is assured that $\mathcal{P}_a$'s long-term public key is authentic and $\mathcal{P}_a$ does know the corresponding private key. In comparison to simply reading out or scanning two fingerprints, a mutual cryptographic authentication that also requires to demonstrate knowledge of the respective private key requires the creation of one additional QR code and thus one additional scanning process. Though we believe that the above solution works it requires some overhead since signatures are not included in TEXTSECURE, yet. Therefore we propose another mitigation in the next section (and analyze it in section IV) that blends well with the cryptographic primitives that are already implemented in TEXTSECURE.

2) Alternative Authentication Method: The challenge response process detailed below can be modified to achieve cryptographic authentication with the primitives already used in TEXTSECURE. The process is as follows:

1) $\mathcal{P}_b$ chooses a keypair $(r, g^r) \in_R \mathbb{Z}_p \times \mathsf{Curve25519}$. It creates a QR code containing $\mathsf{chall} = g^r$.
2) $\mathcal{P}_a$ scans the QR code, derives $\mathsf{resp} = \mathsf{chall}^a$ $(= g^{r \cdot a})$ and creates a QR code containing resp.
3) $\mathcal{P}_b$ scans the QR code presented by $\mathcal{P}_a$ and checks if $g^{a \cdot r} =^? \mathsf{resp}$.

If resp, matches $g^{a \cdot r}$, $\mathcal{P}_b$ is assured that $\mathcal{P}_a$ knows the private key corresponding to the long-term public key that $\mathcal{P}_b$ expected to belong to $\mathcal{P}_a$. If $\mathsf{resp} \neq g^{b \cdot r}$, the authentication fails. Here, we call $\mathcal{P}_a$ prover and $\mathcal{P}_b$ verifier. The process can then be repeated with reversed roles to achieve mutual authentication of $\mathcal{P}_b$ and $\mathcal{P}_a$. We discuss the security of this approach in detail in the following section.

Remark 3. *Both of the proposed protocols rely on QR codes as transport medium, such that a man-in-the-middle attack is hard to conduct unnoticed, as the respective protocols are carried out when participating parties meet in person. Each participant of the authentication process can thus confirm who the other participant actually is and may refuse to continue. As mentioned in Remark 1, this only holds if no two parties collaborate.*

IV. Security of TextSecure Key Exchange and Messaging

As mentioned in Section III, the underlying problem that allows for our attacks is that the shared key between two parties is not cryptographically authentic. In the same section we explain how to face this problem. In this section we first show that the method proposed in Section III-F2 actually solves this issue (under some reasonable restrictions). Next, we show that if public keys are authentic then so is $k = (\mathsf{k}_{\mathsf{Enc}}, \mathsf{k}_{\mathsf{MAC}})$. Moreover we show k to be uniformly distributed. Once we have established this, we finally show that the encryption block of TextSecure is actually one-time stateful authenticated encryption (a primitive which needs uniformly distributed and authenticated keys to provide security).

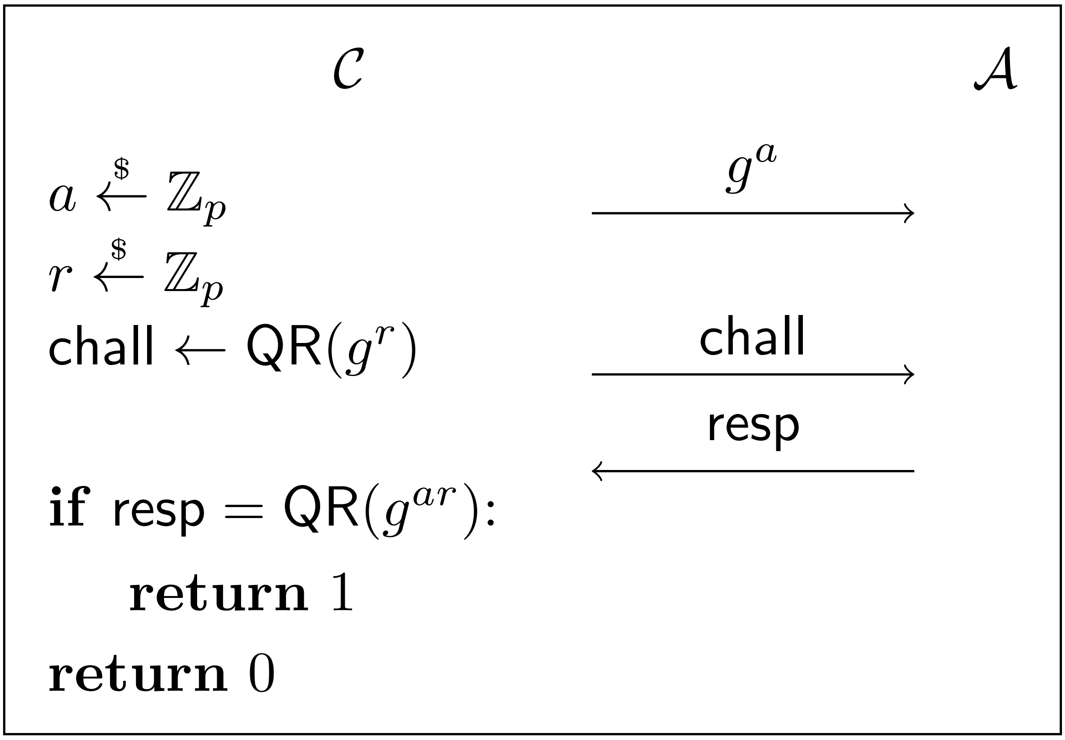

$\mathcal{C}$
$\mathcal{A}$
$a \overset{\$}{\leftarrow} \mathbb{Z}_p$
g^a
$r \overset{\$}{\leftarrow} \mathbb{Z}_p$
$\mathsf{chall} \leftarrow \mathsf{QR}(g^r)$
chall
resp
if $\mathsf{resp} = \mathsf{QR}(g^{ar})$:
return 1
return 0

Figure 6: Challenge and Response security game.

A. *Offline Verification gives authenticity*

In this section, we prove the proposed protocol of Section III-F2 to be secure. Ideally, we could prove the possession of the secret key through a *zero knowledge proof of knowledge*. However, we do not know if the protocol satisfies this strong property. Rather we prove that if an “honest” verifier accepts, then with high probability the prover’s public key is unique. Since the public key is later on used in the means of authentication, this gives authenticity. Note that we do not consider collusion of parties that deliberately agree to use the same long-term key pair here. In this case, attacks as above are always possible. We assume in the following that no two parties collude.

Consider the security game that is depicted in Figure 6. We say that an attacker (t, ϵ)-wins the security game if it runs in time t and $\mathcal{C}$ outputs 1 with probability at least ϵ. We argue that this security game actually models malicious behaviour of an adversary convincing an “honest” verifier to have public key g^a. To this end, suppose that $\mathcal{A}$ publishes the public key g^a that is already registered as the public key of $\mathcal{P}_a$ as its own public key. If $\mathcal{C}$ returns 1, then $\mathcal{A}$ is obviously successful in convincing a party that follows the protocol honestly that g^a is authentic with respect to $\mathcal{A}$ (which it is not). However, if the probability that $\mathcal{A}$ succeeds in the security game is *small* then this is not very likely to happen which means that g^a is registered only once with overwhelming probability[3]. We note that we do not allow the adversary to challenge back the challenger. This somewhat artificial restriction is due to the fact that we cannot prove the scheme

[3] Observe that the probability that two parties following the protocol honestly publish the same public key is $\frac{1}{|\mathsf{Curve25519}|}$

to be secure without this requirement (see below). We remark, however, that this protocol is carried out when prover and verifier meet face-to-face and, moreover, that no data is sent through the data channel. Thus, the adversary is not a network attacker and in particular has not the capability to read, replay, delay, alter, or drop any data that is exchanged throughout a protocol run between two honest parties. A proving party will always be able to confirm who the verifier actually is and may refuse to prove something to a seemingly malicious verifier.

Technically, if we wanted to get rid of this requirement we could use an IND-CCA-secure KEM that supports public keys of the form g^a in elliptic curves, e.g. the PSEC-KEM that is standardized by the ISO [31] and proven to be secure by Shoup [32] in the random oracle model.

Claim 1. *If there is an attacker $\mathcal{A}$ that (t, ϵ)-wins the above game (cf. Figure 6) then there is an algorithm $\mathcal{B}$ that (t', ϵ')-breaks the* DDH *assumption in* Curve25519 *where $t \approx t'$ and $\epsilon \leq \epsilon'$.*

Proof: Suppose $\mathcal{A}$ wins the game with probability ϵ. We construct a DDH-distinguisher $\mathcal{B}$ that runs $\mathcal{A}$ as a subroutine. $\mathcal{B}$ gets as input (g, g^a, g^b, g^γ) and wants to distinguish whether $\gamma = ab$ or not. $\mathcal{B}$ simulates $\mathcal{C}$ as follows: It creates a QR code containing g^b. If $\mathcal{A}$ creates a QR code containing g^γ then $\gamma = ab$ and thus $\mathcal{B}$ is able to solve DDH. ■

B. $(\mathsf{k}_{\mathsf{Enc}}, \mathsf{k}_{\mathsf{MAC}})$ *is authentic.*

Now let us assume public keys are unique. We argue that in this case $k = (\mathsf{k}_{\mathsf{Enc}}, \mathsf{k}_{\mathsf{MAC}})$ authenticates sender and receiver since these are the only parties to compute k.
Here, we show that the sender ($\mathcal{P}_a$) is authenticated. The proof for the receiver is similar. To this end, we define the following algorithm that reflects key derivation (for the first message sent during a session) on the side of the receiver ($\mathcal{P}_b$) where g^a, $g^{\bar{x}_{a,0}}$ and $g^{\bar{x}_{a,2}}$ are long-term and ephemeral public keys of $\mathcal{P}_a$, b is the long-term secret of $\mathcal{P}_b$ and z is a pointer to the prekey pair $(x_{b,z}, g^{x_{b,z}})$ (cf. Figure 3).

Algorithm 2 $\mathsf{Key.derive}\left(g^a, g^{\bar{x}_{a,0}}, g^{\bar{x}_{a,2}}, b, z\right)$

```
$\mathsf{secret} \leftarrow f\left(g^{x_{b,z} \cdot a}, g^{b \cdot \bar{x}_{a,0}}, g^{x_{b,z} \cdot \bar{x}_{a,0}}\right)$
$(k_{BA,r}, k_{BA,c}) \leftarrow f\left(\mathsf{secret}, \mathsf{const}_0, \mathsf{const}_R\right)$
$k_{\text{shared}} \leftarrow g^{\bar{x}_{a,2} \cdot x_{b,z}}$
$(k_{AB,r}, k_{AB,c}) \leftarrow f\left(k_{\text{shared}}, k_{AB,r}, \mathsf{const}_R\right)$
$k \leftarrow f\left(MAC(k_{AB,c}, \mathsf{const}_1), \mathsf{const}_0, \mathsf{const}_K\right)$
```

Let us now consider the security game that is depicted in Figure 7. We say that $\mathcal{A}$ (t, ϵ)-wins the security game if it runs in time at most t and $\mathcal{C}$ returns 1 with probability at least ϵ. We argue that this security game acutally models authenticity of k. To this end, suppose again that g^a is the public key of $\mathcal{P}_a \neq \mathcal{A}$. Note that this public key is unique. It is given to $\mathcal{A}$, together with the public key g^b and a prekey $g^{x_{b,z}}$ of party $\mathcal{P}_b$ by computing k. The goal of $\mathcal{A}$ is to impersonate $\mathcal{A}$ to $\mathcal{P}_b$. If $\mathcal{A}$ is able to succeed, then it may obviously break the authenticity property. On the other hand, if the probability for $\mathcal{A}$ to succeed is small it is very unlikely for $\mathcal{A}$ to compute the shared key of two honest parties $\mathcal{P}_a$ and $\mathcal{P}_b$. By the uniqueness of public keys this gives authenticity.

Claim 2. *If an attacker $\mathcal{A}$ (t, ϵ)-wins the above security game (cf. figure 7), then there is an algorithm $\mathcal{B}$ that (t', ϵ')-breaks the* CDH*-assumption in* Curve25519 *where $t \approx t'$ and $\epsilon \leq \epsilon' + \frac{t'}{2^{512}}$. The analysis will view the functions f and MAC as a* non-programmable *random oracle [33].*

Remark 4. *Following Krawczyk [34], for the proof of claim 2 we resort to a random oracle.*

Proof: We describe a CDH-forger $\mathcal{B}$ that runs $\mathcal{A}$ as a subroutine. $\mathcal{B}$ gets as input $(g, g^a, g^{x_{b,z}})$ and wants to compute $g^{a \cdot x_{b,z}}$. It samples $b \xleftarrow{\$} \mathbb{Z}_p$ and sends $(g^a, g^b, g^{x_{b,z}})$ to $\mathcal{A}$. Note that $\mathcal{B}$ is not able to compute Key.derive. Instead we let $\mathcal{B}$ always return 0. We argue that with overwhelming probability this will not be detected by $\mathcal{A}$: We observe that since f is modeled as a random oracle (and thus the image of f is uniformly distributed over $\{0,1\}^{512}$) for $\mathcal{A}$ to tell the value of k it needs to query f on $\mathrm{MAC}_{k_{AB,c}}((\mathsf{const}_1), \mathsf{const}_0, \mathsf{const}_K)$ since otherwise the value of k is information-theoretically hidden from $\mathcal{A}$. The same argument applies for the value of $\mathrm{MAC}_{k_{AB,c}}$ (which is needed to compute k), $k_{BA,r}$ (which is needed to compute $k_{AB,c}$) and secret (which is needed to compute $k_{BA,r}$). Now, suppose $\mathcal{A}$ queries f on secret $= (g^{a \cdot x_{b,z}}, g^{b \cdot \bar{x}_{a,0}}, g^{x_{b,z} \cdot \bar{x}_{a,0}})$ for some $\bar{x}_{a,0}$. Since f is modeled as a random oracle, the query of $\mathcal{A}$ is actually public and thus $\mathcal{B}$ can extract $g^{a \cdot x_{b,z}}$, the solution to the CDH instance. Thus, if CDH is hard in Curve25519 for

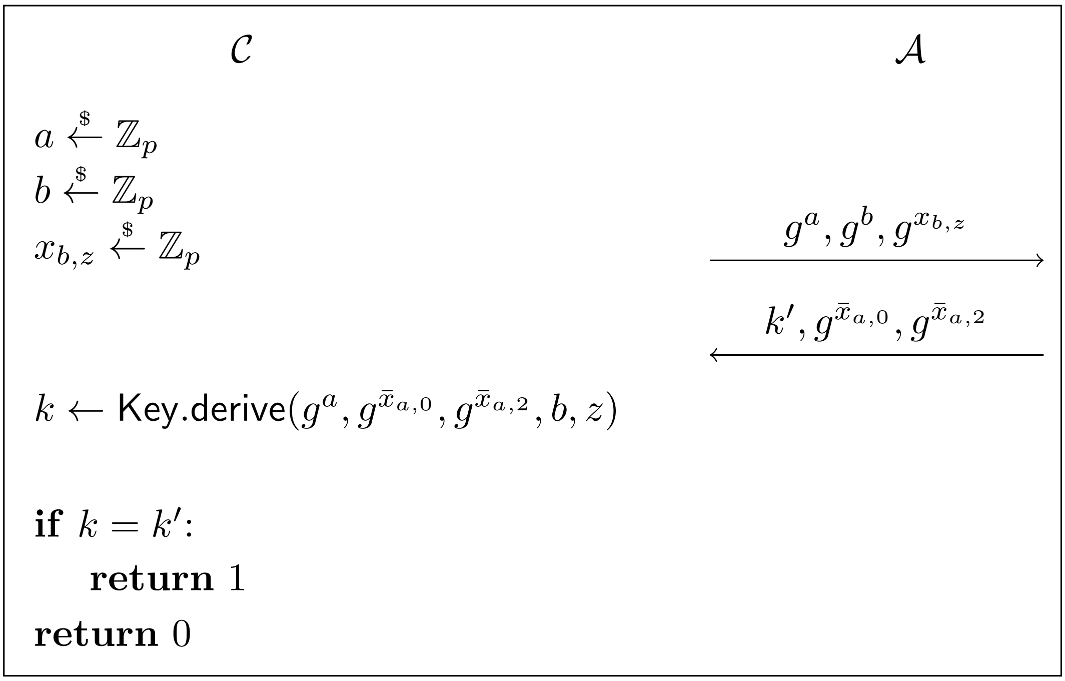

$\mathcal{C}$
$\mathcal{A}$
$a \xleftarrow{\$} \mathbb{Z}_p$
$b \xleftarrow{\$} \mathbb{Z}_p$
$x_{b,z} \xleftarrow{\$} \mathbb{Z}_p$
$g^a, g^b, g^{x_{b,z}}$
$k', g^{\bar{x}_{a,0}}, g^{\bar{x}_{a,2}}$
$k \leftarrow \mathsf{Key.derive}(g^a, g^{\bar{x}_{a,0}}, g^{\bar{x}_{a,2}}, b, z)$
if $k = k'$:
return 1
return 0

Figure 7: Authenticity of k security game.

$\mathcal{A}$, to be successful, $\mathcal{A}$ needs to correctly guess a bitstring of length 512. This probability can be neglected. ■

We immediately obtain that k is not only authentic but also uniformly distributed. Now, a similar (information theoretic) argument applies to keys that are derived from k for the next messages to be sent and received.

We stress that for our proof to be bootstrapped to the setting with more than one prekey of $\mathcal{P}_b$, these have to be pairwise distinct (i.e., unique) since otherwise replay attacks become possible. In particular, Claim 2 does not apply to keys exchanged relying on the *last resort key*.

C. TEXTSECURE *Encryption is One-time Stateful and Authenticated*

Next, we prove the actual encryption of TEXTSECURE to be one-time stateful authenticated encryption. We stress that one-time security suffices for TEXTSECURE since, here, the actual encryption and MAC keys are updated (and can be seen as fresh, cf. previous section) with every message to be sent.

1) Cryptographic Primitives: We shortly recall the cryptographic primitives that are used by the TEXTSECURE encryption procedure.

A *message authentication code* is a pair of PPT algorithms $\mathsf{MAC} = (\mathsf{Tag}, \mathsf{Vfy})$ such that $\mathsf{tag} \xleftarrow{\$} \mathsf{Tag}(k, m)$ on input a key k and a message m returns tag for that message. The algorithm $\{0,1\} \leftarrow \mathsf{Vfy}(k, m, \mathsf{tag})$ returns $\mathsf{Tag}(k, m) \overset{?}{=} \mathsf{tag}$. We require the usual correctness properties. Following Bellare et al. [35], [36] we say that an attacker $\mathcal{A}$ (t, ϵ)-breaks the strong one-time security of MAC if it runs in time t and

$$\Pr\left[(\mathsf{tag}, m) \xleftarrow{\$} \mathcal{A}^{\mathsf{Tag}(k,)} : \begin{array}{c}\mathsf{Vfy}(k, m, \mathsf{tag}) = 1 \\ \wedge\, (\mathsf{tag}, m) \neq (\mathsf{tag}', m')\end{array}\right] \geq \epsilon$$

where $\mathcal{A}$ is allowed to query Tag at most one time (the query is denoted by m' and the response by tag').

A *symmetric encryption scheme* is a pair of PPT algorithms $\mathsf{SE} = (\mathsf{Enc}, \mathsf{Dec})$ such that $c \xleftarrow{\$} \mathsf{Enc}(k, m)$ on input a key k and a message m returns a ciphertext c and $m \leftarrow \mathsf{Dec}(k, c)$ on input a key k and a ciphertext c outputs m. We require the usual correctness properties. We say that an attacker $\mathcal{A}$ (t, ϵ)-breaks the one-time IND-CPA-security [37] of SE if it runs in time t and

$$\mathsf{Pr}\left[b' \xleftarrow{\$} \mathcal{A}^{\mathsf{Encrypt}(\cdot,\cdot)} : b = b'\right] \geq \epsilon$$

where $\mathcal{A}$ is allowed to query Encrypt at most one time on two messages m_0 and m_1 of equal length and Encrypt samples a uniformly random bit b and returns $c \xleftarrow{\$} \mathsf{Enc}(k, m_b)$.

2) Stateful Authenticated Encryption in TEXTSECURE*:* A stateful encryption scheme Ste is a pair of PPT algorithms Ste = (Ste.Enc, Ste.Dec) with the following syntax:

- The encryption algorithm outputs a ciphertext C and a state st' on input key k, associated data hd, message m and a state st, $(C, st') \xleftarrow{\$} \mathsf{Ste.Enc}(k, hd, m, st)$.
- The deterministic decryption algorithm outputs a message m and a state st' on input a key k, associated data hd, a ciphertext C and a state st, $(m, st') \xleftarrow{\$} \mathsf{Ste.Dec}(k, hd, C, st)$. Ste.Dec may also return $\perp$ indicating a decryption error.

For TEXTSECURE we let hd contain at least $(g^{\bar{x}_{a,2}}, \mathsf{regID}_a, \mathsf{regID}_b)$, the ephemeral public key of party $\mathcal{P}_a$, and the identifiers of both parties. The state is set to $(\mathsf{ctr}, \mathsf{pctr})$ and is initialized to $(0, 0)$. For practical purposes it is important that these are handed over carefully to the next execution of encrypt and decrypt, resp. The key is set to be $k = (\mathsf{k}_{\mathsf{Enc}}, \mathsf{k}_{\mathsf{MAC}})$ handed over by the ratchet protocol.

Algorithm 3 $\mathsf{TS.Enc}(k, hd, m, st)$

```
$c \leftarrow \mathsf{Enc}_{\mathsf{k}_{\mathsf{Enc}}}(m)$
$\chi \leftarrow (v, g^{\bar{x}_{a,2}}, \mathsf{ctr}_a, \mathsf{pctr}_a, c)$
$\mathsf{tag} \leftarrow \mathsf{MAC}_{\mathsf{k}_{\mathsf{MAC}}}(\chi)$
$\mathsf{ctr}++$
return $(\underbrace{\chi, \mathsf{tag}}_{\mathbf{C}}, st)$
```

Algorithm 4 $\mathsf{TS.Dec}(k, hd, \mathbf{C}, st)$

```
parse $\mathbf{C}$ as $\mathbf{C} = (\chi', \mathsf{tag}')$
parse $\chi'$ as $\chi' = (v', (g^{\bar{x}_{a,2}})', \mathsf{ctr}'_a, \mathsf{pctr}'_a, c')$
$\chi'' \leftarrow (v', (g^{\bar{x}_{a,2}})', \mathsf{pctr}_b, \mathsf{ctr}_b, c')$
$\mathsf{tag}'' \leftarrow \mathsf{MAC}_{\mathsf{k}_{\mathsf{MAC}}}(\chi'')$
if $\mathsf{tag}'' \neq \mathsf{tag}'$ return $(\perp, \perp)$
$m \leftarrow \mathsf{Dec}_{\mathsf{k}_{\mathsf{Enc}}}(c)$
if $m = \perp$ return $(\perp, \perp)$
$\mathsf{pctr}++$
return $(m, st)$
```

Security of a Stateful Authenticated Encryption scheme Ste is captured through a security game that is played between a challenger $\mathcal{C}$ and an attacker $\mathcal{A}$, as implicitly mentioned by Paterson et al. [18] and explicitly by Jager et al. [38].

- The challenger samples $b \xleftarrow{\$} \{0, 1\}$ and a key $k \xleftarrow{\$} \mathcal{K}$. It initializes states and sets $i = \mathsf{ctr} - 1$ and $j = \mathsf{pctr} - 1$ (This way it is guaranteed that $C_i = C_{\mathsf{ctr}_a}$ whenever a ciphertext is generated by a and similar for b).
- The adversary may then query each of the Encrypt and Decrypt oracles once and they respond as depicted in Figure 8.
- Finally, $\mathcal{A}$ outputs a bit b' and wins if $b = b'$.

```
Encrypt$(m_0, m_1, hd)$
$i = i + 1$
$(C^0, st^0) \xleftarrow{\$} \mathsf{Ste.Enc}(k, hd, m_0, st)$
$(C^1, st^1) \xleftarrow{\$} \mathsf{Ste.Enc}(k, hd, m_1, st)$
if $C^0 = \bot \vee C^1 = \bot$: return $\bot$
$(C_{\mathsf{ctr}}, st_e) \leftarrow (C^b, st^b)$
return $C_i$
```

```
Decrypt$(C, hd)$
$j = j + 1$
$(m, st) \leftarrow \mathsf{Ste.Dec}(k, hd, C, st)$
if $j > i \vee C \neq C_j$ : $\mathsf{div} \leftarrow 1$
if $\mathsf{div} = 1 \wedge b = 1$ return m
return $\bot$
```

Figure 8: Encrypt and Decrypt Oracles in the Stateful Authenticated Encryption security game.

Since the behaviour of oracle Decrypt might not be clear at first sight, we will shortly explain it. First, we recall that the goal of the adversary is to determine the bit b. Stated otherwise, if the adversary queries Encrypt on two *distinct* messages of equal length its goal is to determine which message is encrypted. Oracle Decrypt will reveal this information to $\mathcal{A}$ if either $\mathcal{A}$ manages to query Decrypt for a ciphertext that is not authentic, i.e., $C \neq C_j$ or that is in the wrong order, i.e., $j > i$ and that nonetheless does not incur in a decryption error.

Theorem 1. *If there is an attacker $\mathcal{A}$ that (t, ϵ)-breaks the one-time stateful authenticated encryption security of* TS.Ste *then there is an attacker $\mathcal{B}_{\mathsf{MAC}}$ that (t_1, ϵ_1)-breaks the strong one-time security of* MAC *and an attacker $\mathcal{B}_{\mathsf{Enc}}$ that (t_2, ϵ_2)-breaks the one-time* IND-CPA *security of the encryption scheme where $t \approx t_1 \approx t_2$ and*

$$\epsilon \leq \epsilon_1 + \epsilon_2$$

Proof: The proof is by a sequence of games. First, we will modify the Decrypt oracle such that its response will be independent of b. If the MAC scheme is secure this change will be undetected by $\mathcal{A}$. After that we will rely on the security of Enc to argue that $\mathcal{A}$ has only negligible advantage in telling the value of b. By ζ_i we will denote the event that $\mathcal{A}$ is successful in Game i.

GAME 0. This is the real Stateful Authenticated security game as described above. Thus, we have:

$$\mathsf{Pr}[\zeta_0] = \epsilon$$

GAME 1. In Game 1, the challenger proceeds as follows: When the attacker queries a ciphertext C to the decryption oracle, $\mathcal{C}$ always outputs $\bot$. Except for this, $\mathcal{C}$ proceeds exactly as in Game 0. Note that Decrypt will return always $\bot$ if $\mathsf{div} = 0$ (i.e., conditioned on $\mathsf{div} = 0$ Game 0 and Game 1 are identical) and will return m only if $\mathsf{div} = b = 1$. Now, if $\mathsf{div} = 1$ then $j > i \vee C \neq C_i$.

$j > i$: If $j > i$ (i.e. Decrypt is called by $\mathcal{A}$ before Encrypt) and decryption does not return $\bot$ then the ciphertext C that was queried to Decrypt by the adversary contains a valid tag which contradicts the one-time security of MAC.

$C \neq C_j$: If $C \neq C_j$ then a similar argument applies to show that decryption will fail and thus return $\bot$: Here, we reduce to the *strong* one-time security of MAC. (Recall that strong one-time security requires that it is computationally infeasible to compute a valid tag, tag', for message m, even if a valid tag, tag, for message m is known.Thus Thus, we have:

$$\mathsf{Pr}[\zeta_0] - \mathsf{Pr}[\zeta_1] \leq \epsilon_1$$

GAME 2. In Game 2, instead of encrypting m_b, $\mathcal{C}$ samples a random message m, computes $(C, st) \xleftarrow{\$} \mathsf{TS.Enc}(k, hd, m, st)$ and returns C. This change does not affect oracle Decrypt since due to Game 1 it will return $\bot$ anyway. Now, by the IND-CPA-security of Enc this goes unnoticed from the adversary. Thus, we have:

$$\mathsf{Pr}[\zeta_1] - \mathsf{Pr}[\zeta_2] \leq \epsilon_2$$

Claim 3. $\mathsf{Pr}[\zeta_2] = 0$.

Proof: In Game 2 the Decrypt oracle reveals no information about b due to Game 1. Neither does the Encrypt-oracle due to Game 2. The claim follows. ■

D. Summary

In this section, we showed that our improvement of the TEXTSECURE protocol mitigates the UKS attacks. Therefore we showed, that the protocol from section III-F2 proves long-term public keys to be unique, i.e., that with overwhelming probability no two parties share the same public key (except if they collude). Then we proved that $(\mathsf{k}_{\mathsf{Enc}}, \mathsf{k}_{\mathsf{MAC}})$ are bound to the respective long-term keys of the parties which gives authentic keys (that are also uniformly distributed). Finally, we established that TEXTSECURE encryption is actually one-time stateful and authenticated. If the encryption and decryption states are carried over carefully from one call of the encryption and decryption procedure to the next, this shows that TEXTSECURE achieves stateful authenticated encryption.

V. Related Work

The body of work that explicitly aims at providing or analyzing secure instant messaging protocols is comparably small to the prevalence of instant messaging applications in our daily life: At the 2004 Workshop on Privacy in the Electronic Society (WPES), Borisv et. al. [3] presented a protocol for “Off the Record” (OTR) communication. The OTR protocol was designed to provide authenticated and confidential instant messaging communication with strong perfect forward secrecy and *deniability*: no party can cryptographically prove the authorship of a message. The deniability property of OTR has been discussed by Kopf and Brehm [39]. The work of Di Raimondo et. al. [30], who analyzed the security of OTR, is in its nature closely related to our paper. The authors point out several issues with OTR’s authentication mechanism and also describe a UKS attack on OTR, as well as, a replay attack along with fixes. We note, however, that the authentication mechanisms of OTR and TEXTSECURE have little in common: Though it aims to provide deniability, OTR explicitly uses signatures for authentication while TEXTSECURE does not. The UKS attack on OTR described by Raimondo et al. [30] directly targets the key exchange mechanism of the protocol, whereas the attacks presented in this paper are rather subtle and exploit the protocol structure and key derivation of TEXTSECURE.

Besides OTR, which has been widely adopted, there exist protocols for secure instant messaging like IMKE [40], which aim at being verifiable in a BAN-like logic, but has never found a wider adoption. SILC [41] also has received a certain adoption and some discussion, but is rarely used today, as is FiSH [42], a once popular plugin for IRC clients that used Blowfish with pre-shared keys to encrypt messages.

Thomas [43] analysed the browser-based instant messager Cryptocat and found that, due to an implementation error, Cryptocat used private keys of unsufficient length when establishing a group chat session. Green [44] recently discussed Cryptocat’s group chat approach from a protocol perspective and points out several issues.

Further protocols exist that aim at securing instant messaging communication but have, to the best of our knowledge, not received public scrutiny. Among these are THREEMA [45], SURESPOT [46], and Silent Circle’s SCIMP [47].

VI. Conclusion and Future Work

Since Facebook bought WhatsApp, instant messanging apps with security guarantees became more and more popular. The most prominent mobile applications for *secure* IM are THREEMA [45], SURESPOT [46], and TEXTSECURE. In this paper, we have provided a detailed security analysis of TEXTSECURE. First, we precisely described the protocol and then performed a security analysis of the individual steps of the protocol. This led to the discovery of several peculiarities, most notably the protocol’s susceptibility for a UKS attack. We proposed a mitigation and proved that, if our mitigation is applied, that TEXTSECURE actually provides stateful authenticated encryption. To the best of our knowledge, this is the first formal verification of the security guarantees offered by the tool.

While TEXTSECURE’s implementation is open source, little is known about the competing messaging applications. While THREEMA makes use of the open source libary NaCl [48] for cryptographic operations, its protocol is kept confidential. SURESPOT is an open source project with its own cryptographic protocol; an analysis of its protocol has (as far as we know) not been done yet. A comparison of TEXTSECURE and SURESPOT will be an interesting project for future work.

ACKNOWLEDGEMENT

In a previous version of the paper, we described what we considered a unconventional use of HMAC, as at one point of the protocol, HMAC is used with const_0 as key and $(\mathsf{secret}, \mathsf{const}_R)$ as input. However, based on Dodis et al. [49], Krawczyk has shown the security of this application of HMAC in the Random Oracle Model [34]. The authors thank David L. Gil highlighting the aspect, Brian Smith for pointing out several publications, and Trevor Perrin and Moxie Marlinspike, as well as, Tibor Jager for their comments and insightful discussion.

REFERENCES

[1] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, "OpenPGP Message Format," RFC 4880 (Proposed Standard), Internet Engineering Task Force, Nov. 2007, updated by RFC 5581. [Online]. Available: http://www.ietf.org/rfc/rfc4880.txt

[2] B. Ramsdell and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification," RFC 5751 (Proposed Standard), Internet Engineering Task Force, Jan. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5751.txt

[3] N. Borisov, I. Goldberg, and E. A. Brewer, "Off-the-record communication, or, why not to use pgp," in *WPES*, 2004, pp. 77–84.

[4] Open WhisperSystems, "The new TextSecure: Privacy beyond SMS," Feb. 2014. [Online]. Available: https://whispersystems.org/blog/the-new-textsecure/

[5] ——, "TextSecure, now with 10 million more users," Dec. 2013. [Online]. Available: https://whispersystems.org/blog/cyanogen-integration/

[6] M. Crider, "CyanogenMod is now installed on over 10 million android devices," Dec. 2013. [Online]. Available: http://www.androidpolice.com/2013/12/22/cyanogenmod-is-now-installed-on-over-10-million-android-devices/

[7] M. Marlinspike, "Internet explorer SSL vulnerability," May 2002. [Online]. Available: http://www.thoughtcrime.org/ie-ssl-chain.txt

[8] ——, "More tricks for defeating SSL in practice," 2009. [Online]. Available: https://www.blackhat.com/presentations/bh-usa-09/MARLINSPIKE/BHUSA09-Marlinspike-DefeatSSL-SLIDES.pdf

[9] ——, "sslstrip," 2011. [Online]. Available: http://www.thoughtcrime.org/software/sslstrip/

[10] ——, "Convergence | beta," 2011. [Online]. Available: http://convergence.io/details.html#

[11] ——, "chapcrack," 2012. [Online]. Available: https://github.com/moxie0/chapcrack

[12] M. Marlinspike and T. Perinn, "Trust assertions for certificate keys draft-perrin-tls-tack-02.txt," 2013. [Online]. Available: http://tack.io/draft.html

[13] "Edward snowden and ACLU at SXSW," Mar. 2014. [Online]. Available: https://www.youtube.com/watch?v=UIhS9aB-qgU&t=14m24s

[14] W. Diffie, P. C. van Oorschot, and M. J. Wiener, "Authentication and authenticated key exchanges," *Des. Codes Cryptography*, 1992.

[15] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *PKC*, 2006. [Online]. Available: http://cr.yp.to/ecdh/curve25519-20060209.pdf

[16] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication," RFC 7235 (Proposed Standard), Internet Engineering Task Force, Jun. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7235.txt

[17] T. Perrin, "Axolotl ratchet," Jun. 2014. [Online]. Available: https://github.com/trevp/axolotl/wiki

[18] K. G. Paterson, T. Ristenpart, and T. Shrimpton, "Tag size does matter: Attacks and proofs for the TLS record protocol," in *Advances in Cryptology – ASIACRYPT 2011*, ser. Lecture Notes in Computer Science, D. H. Lee and X. Wang, Eds., vol. 7073. Seoul, South Korea: Springer, Berlin, Germany, Dec. 4–8, 2011, pp. 372–389.

[19] E. Barker, D. Johnson, and M. Schmid, "Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography (revised)," NIST Special Publication 800-56a, National Institute of Standards and Technology, Tech. Rep., 2007, http://csrc.nist.gov/publications/nistpubs/800-56A/SP800-56A_Revision1_Mar08-2007.pdf.

[20] C. Meyer, J. Somorovsky, E. Weiss, J. Schwenk, S. Schinzel, and E. Tews, "Revisiting ssl/tls implementations: New bleichenbacher side channels and attacks," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014.

[21] A. Karjalainen and N. Mehta, "The heartbleed bug," 2014. [Online]. Available: http://heartbleed123.com/

[22] C. Meyer and J. Schwenk, "Sok: Lessons learned from ssl/tls attacks," in *WISA*, 2013, pp. 189–209.

[23] N. J. AlFardan and K. G. Paterson, "Lucky thirteen: Breaking the tls and dtls record protocols," in *IEEE Symposium on Security and Privacy*, 2013.

[24] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. N. Schuldt, "On the security of rc4 in tls," in *USENIX Security*, 2013.

[25] S. Uellenbeck, M. Drmuth, C. Wolf, and T. Holz, "Quantifying the security of graphical passwords: The case of android unlock patterns," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 161–172. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516700

[26] A. J. Aviv, K. Gibson, E. Mossop, M. Blaze, and J. M. Smith, “Smudge attacks on smartphone touch screens,” in *Usenix WOOT*, 2010.

[27] S. Blake-Wilson and A. Menezes, “Unknown key-share attacks on the station-to-station (STS) protocol,” in *Public Key Cryptography*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Jan. 1999, no. 1560, pp. 154–170. [Online]. Available: http://link.springer.com/chapter/10.1007/3-540-49162-7_12

[28] B. S. Kaliski, Jr., “An unknown key-share attack on the MQV key agreement protocol,” *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 3, pp. 275–288, Aug. 2001. [Online]. Available: http://doi.acm.org/10.1145/501978.501981

[29] C. Boyd and A. Mathuria, *Protocols for Authentication and Key Establishment*. Springer, 2003.

[30] M. D. Raimondo, R. Gennaro, and H. Krawczyk, “Secure off-the-record messaging,” in *WPES*, 2005, pp. 81–89.

[31] ISO/IEC, “Iso/iec 18033-2:2006, information technology – security techniques – encryption algorithms – part 2: Asymmetric ciphers,” International Organization for Standardization, Tech. Rep., 2006.

[32] V. Shoup, “A proposal for an iso standard for public key encryption,” Cryptology ePrint Archive, Report 2001/112, 2001, http://eprint.iacr.org/.

[33] M. Fischlin, A. Lehmann, T. Ristenpart, T. Shrimpton, M. Stam, and S. Tessaro, “Random oracles with(out) programmability,” in *Advances in Cryptology – ASIACRYPT 2010*, ser. Lecture Notes in Computer Science, M. Abe, Ed., vol. 6477. Singapore: Springer, Berlin, Germany, Dec. 5–9, 2010, pp. 303–320.

[34] H. Krawczyk, “Cryptographic extraction and key derivation: The HKDF scheme,” in *Advances in Cryptology – CRYPTO 2010*, ser. Lecture Notes in Computer Science, T. Rabin, Ed., vol. 6223. Santa Barbara, CA, USA: Springer, Berlin, Germany, Aug. 15–19, 2010, pp. 631–648.

[35] M. Bellare, R. Canetti, and H. Krawczyk, “Keying hash functions for message authentication,” in *Advances in Cryptology – CRYPTO’96*, ser. Lecture Notes in Computer Science, N. Koblitz, Ed., vol. 1109. Santa Barbara, CA, USA: Springer, Berlin, Germany, Aug. 18–22, 1996, pp. 1–15.

[36] M. Bellare and C. Namprempre, “Authenticated encryption: Relations among notions and analysis of the generic composition paradigm,” in *Advances in Cryptology – ASIACRYPT 2000*, ser. Lecture Notes in Computer Science, T. Okamoto, Ed., vol. 1976. Kyoto, Japan: Springer, Berlin, Germany, Dec. 3–7, 2000, pp. 531–545.

[37] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, “A concrete security treatment of symmetric encryption,” in *38th Annual Symposium on Foundations of Computer Science*. Miami Beach, Florida: IEEE Computer Society Press, Oct. 19–22, 1997, pp. 394–403.

[38] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, “On the security of TLS-DHE in the standard model,” in *Advances in Cryptology – CRYPTO 2012*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Santa Barbara, CA, USA: Springer, Berlin, Germany, Aug. 19–23, 2012, pp. 273–293.

[39] G. Kopf and B. Brehm, “Phrack magazine: Secure function evaluation vs. deniability in OTR and similar protocols,” Apr. 2012. [Online]. Available: http://phrack.org/issues/68/14.html

[40] M. Mannan and P. C. van Oorschot, “A protocol for secure public instant messaging,” in *Financial Cryptography and Data Security*, 2006.

[41] P. Riikonen, “Secure internet live conferencing protocol specification DRAFT,” 2007. [Online]. Available: http://tools.ietf.org/id/draft-riikonen-silc-spec-09.txt

[42] Unkown, “FiSH – secure communications with internet relay chat,” 2007. [Online]. Available: http://ultrx.net/doc/fish/

[43] S. Thomas, “DecryptoCat,” 2013. [Online]. Available: http://tobtu.com/decryptocat-old.php

[44] M. Green, “Noodling about IM protocols,” Jul. 2014. [Online]. Available: http://blog.cryptographyengineering.com/2014/07/noodling-about-im-protocols.html

[45] Threema, “Seriously secure mobile messaging.” [Online]. Available: https://threema.ch/de/

[46] surespot, “encrypted messenger.” [Online]. Available: https://surespot.me/

[47] V. Moscaritolo, G. Belvin, and P. Zimmermann, “Silent circle instant messaging protocol protocol specification,” 2012.

[48] D. J. Bernstein, T. Lange, and P. Schwabe, “The security impact of a new cryptographic library,” in *Progress in Cryptology - LATINCRYPT 2012: 2nd International Conference on Cryptology and Information Security in Latin America*, ser. Lecture Notes in Computer Science, A. Hevia and G. Neven, Eds., vol. 7533. Santiago, Chile: Springer, Berlin, Germany, Oct. 7–10, 2012, pp. 159–176.

[49] Y. Dodis, R. Gennaro, J. Håstad, H. Krawczyk, and T. Rabin, “Randomness extraction and key derivation using the CBC, cascade and HMAC modes,” in *Advances in Cryptology – CRYPTO 2004*, ser. Lecture Notes in Computer Science, M. Franklin, Ed., vol. 3152. Santa Barbara, CA, USA: Springer, Berlin, Germany, Aug. 15–19, 2004, pp. 494–510.