Post-Compromise Security in Self-Encryption

Gwangbae Choi
Fasoo, Seoul, Korea

F. Betül Durak
Robert Bosch LLC – Research and Technology Center, Pittsburgh PA, USA

Serge Vaudenay
Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

Abstract

In self-encryption, a device encrypts some piece of information for itself to decrypt in the future. We are interested in security of self-encryption when the state occasionally leaks. Applications that use self-encryption include cloud storage, when a client encrypts files to be stored, and in 0-RTT session resumptions, when a server encrypts a resumption key to be kept by the client. Previous works focused on forward security and resistance to replay attacks. In our work, we study post-compromise security (PCS). PCS was achieved in ratcheted instant messaging schemes, at the price of having an inflating state size. An open question was whether state inflation was necessary. In our results, we prove that post-compromise security implies a super-linear state size in terms of the number of active ciphertexts which can still be decrypted. We apply our result to self-encryption for cloud storage, 0-RTT session resumption, and secure messaging. We further show how to construct a secure scheme matching our bound on the state size up to a constant factor.

2012 ACM Subject Classification Security and privacy → Cryptography

Keywords and phrases Encryption, Ratchet, Post-Compromise Security, Instant Messaging, Session Resumption, Cloud Storage

Digital Object Identifier 10.4230/LIPIcs.ITC.2021.25

Acknowledgements We deepy thank the anonymous reviewers for their valuable comments.

1 Introduction

In many deployed applications, the design of the application involves various devices communicating with each other securely. They sometimes require one of the devices to encrypt some piece of information that will be used in the future by itself. We call this *self-encryption*. One application is massive client-server connections where millions of clients connect to a server, causing the server being unable to afford to store any client-specific information. On the other hand, recent protocols such as TLS 1.3 offers an alternate way to make the server to resume past sessions without going through a new round-trip handshake when a client reconnects to the server.[1] While clients would surely benefit from a smooth connection experience, the server has to "remember" each session in a secure manner, possibly by keeping a (small or big) size of state. More precisely, when a client connects to a website for the first time, the web server generates a ticket for the client. This ticket is a piece of information that helps the server to remember the session. Somehow, this is a helper that the server encrypts for itself which is to be kept by the client like cookies. When the client reconnects to the same website with her ticket, the server may use the information contained in the ticket to resume their session. As desired, it gives the freedom not to store any client-specific information on the server-side. However, the server needs a secret state for

[1] As of November 2019, 34% of TLS connections use session resumption [11].

the cryptographic operations which are used in generating and decrypting tickets. From the security point of view, then, the concern becomes to provide security against replay attacks or occasional exposures of the internal state of the server.

In general, the internal state is any type of information that would let a device decrypt (some part of) the communication. In this work, we investigate the security of self-encryption which comes in two forms: forward security (FS) and post-compromise security (PCS). Intuitively, forward security provides security for the *past* communication when exposure happens, whereas post-compromise security aims to heal the *future* communication when exposure occurs [6]. Before going forward with security, we list three applications.

0-RTT in TLS 1.3

In the TLS 1.3 protocol, a client connects to a server and establishes a common secret key through a handshake key agreement protocol. This is succeeded with a full round trip time (1-RTT) communication. Ideally, when the client reconnects to the same server after a while, the connection should be resumed with no round trip time (0-RTT). 0-RTT has been an active research domain in the last few years [2, 7, 10]. It is achieved in practice through two elementary approaches called *session caches* and *session tickets* as described by Aviram, Gellert, and Jager (AGJ) [2]. In the former technique, the server resumes the session by assigning a different resumption key for each connection and sending the client a look-up index that links to the resumption key. The ticket is that index. When the client comes back, it includes the ticket and the payload data. This provides forward security. Nevertheless, the solution depends on maintaining a big database on the server, which is not alluring.

The other approach for 0-RTT in TLS 1.3 configurations is to create session tickets for each client by using a long-term secret key K (the ticket encryption key). Therefore, instead of storing a unique key for each session, the server generates a secret material for each client and encrypts it under K. The secret material is called resumption key whereas the encrypted resumption key is the ticket. The client stores both the resumption key and the ticket. Later on, the client encrypts the payload with the resumption key and includes her ticket in 0-RTT message to remind herself. The server can decrypt the ticket with K and retrieve the resumption secret to decrypt the payload. This approach avoids storing a big database; it is easy to implement and to integrate in existing systems, yet, it does not provide any kind of security in the case of a key exposure.[2]

In their recent work, Aviram, Gellert, and Jager (AGJ) [2] studied the forward security and the resistance to replay attacks of session resumption, specifically focusing on session tickets. However, they did not consider PCS in their security model.

Cloud Storage

In a single client-server cloud storage, the client wants to outsource her files in a remote storage (cloud) in an encrypted form. The encryption of the files occurs locally on the client who keeps the secret decryption material. The adversary has full access to the cloud and can also keep archives of removed storage. If the client encrypts all files with the same key, the leakage of the key becomes catastrophic as all files (even the removed ones) become compromised. Besides, the client aims to minimize the storage on her local while maintaining

[2] In TLS 1.3, it is considered good practice to rotate the long-term key K every few hours by assuming that all the clients will resume their sessions in the "life-time" of K. Nevertheless, as soon as the key K is compromised, there is neither FS nor PCS during the active period of K.

strong security in case of a compromise of her internal state. This cloud storage problem shares similarities with the 0-RTT problem: the cloud client and the 0-RTT server want to minimize their storage while conserving security.

On the other hand, keys should not be used more than what the encryption method can guarantee to be secure or age too long. This is part of a common good practice in key management. Regulations actually mandate the encrypted files to be updated from an old key to a new key often enough. This is called *key rotation*. The fundamental motivation, however, comes with the desire to achieve resilience to key exposure. Key rotation was formally studied by Boneh et al. [3]. More recently, Everspaugh et al. [9] considered the integrity problem with key rotation.

The naive way to achieve key rotation is to make the client download the encrypted files on the local, decrypt them with the existing key, generate a new fresh key, re-encrypt, and finally outsource back. However, it is a very cumbersome solution for the client. The main task of key rotation is to avoid the complexity of communication and the complexity of treatment on the client side. In practice, AWS and Google deploy a more practical methods based on hybrid encryption: a header $\mathsf{ct}_1 = \mathsf{Enc}_K(\mathsf{eph})$ is formed by encrypting an ephemeral key eph and the rest of the ciphertext $\mathsf{ct}_2 = \mathsf{Enc}_{\mathsf{eph}}(\mathsf{pt})$ is formed by encrypting the plaintext pt using eph. Key rotation is done by updating the header as $\mathsf{ct}'_1 = \mathsf{Enc}_{K'}(\mathsf{eph})$ but keeping the same ephemeral key so $\mathsf{ct}'_2 = \mathsf{ct}_2$. This was argued to be a bit cheating with the concept of key rotation as the encryption of data under the same key was remaining in ct_2.

We tackle the privacy problem differently. Instead of updating a ciphertext to be decryptable with a chosen key, we let ciphertexts unchanged but update the state which is stored by the client[3]. Naturally, our concern becomes more focused on the storage space on the client side. In our setting, the client stores one state and needs no operation on ciphertexts.

Instant Messaging

Post-compromise security in instant messaging was formally studied during the last few years [14, 12, 13, 1, 8]. It is addressed by the notion of *ratchet*. A ratchet consists of updating a key in a one-way manner (for FS) by using some unpredictable randomness (for PCS). Bidirectional secure communication applications can be transformed into self-encryption. In fact, roughly speaking, we can merge both participants into one single device which would encrypt for itself. A *ratcheted* scheme is normally FS and PCS secure, hence defines an FS and PCS secure self encryption which we call a *self-ratchet*.

Our Perspective

In order to study the security of self-encryption, we consider a scheme which generates ciphertexts with the ability to decrypt later, even when the state to decrypt evolves. We define it in a way that it covers the three (and potentially more) applications we described earlier. Furthermore, we are interested in forward security and post-compromise security of these systems. The former captures that the system generates ciphertexts that should remain decryptable for a limited time and that are not going to be decryptable anymore after they "expire" (it could happen either because the settings allow the ciphertexts to stay alive for a limited time or because there is an inherent latency to rotate keys). The ciphertexts that are still decryptable are called *active ciphertexts*. Making a ciphertext become inactive

[3] We do not mean to pick a fresh key to "rotate" the key and update the header as practiced by AWS.

is a way to have forward security: if the state of the scheme is exposed after a ciphertext becomes inactive, this ciphertext is still safe. The PCS defines what happens to the security *after* an exposure of a state. When an exposure takes place, the post-compromise secure system should be able to heal the state such that the ciphertexts which are generated after the healing are secure. In many studies, PCS is interchangeably used with *healing*.

While studying self-ratcheted schemes with PCS guarantees (as well as FS), it was intuitive to expect that the state size of any post-compromise secure self-ratcheted scheme will grow because decryption keys would need to be independent. However, it was not clear why and with what bounds we could achieve it. The first contribution of our work is to show that we cannot achieve post-compromise security better than adding a trivial solution to already existing efficient FS schemes.

As for forward security, AGJ [2] specifically consider the session resumption in TLS 1.3 and they designed solutions for FS and replay attacks without PCS. Their construction is practical. In another study by Günther et al. [10] and Derler et al. [7], the authors consider a solution without any shared secret. In these works, the clients resume connections without having to store any session-specific information on her local. The client keeps only the long-term public key pk of the server. Therefore, they look for forward-secure solutions when the long-term secret key sk evolves throughout time although the associated public key never changes, hence the clients never updates its state. Although it is remarkable that such schemes with forward security exist, both constructions are less practical due to the heavy cryptographic tools they use. Therefore, we rather focus on the FS scheme AGJ to add PCS.

In their seminal paper on PCS, Cohn-Gordon, Cremers, and Garratt [6] focus on Authenticated Key Exchange (AKE). In AKE, the protocol starts with a state and ends when both participants have obtained the exchanged key. The typical exposure threats happen before or after the protocol but not during it because the protocol is rather short. The AKE protocol proposed by Cohn-Gordon et al. [6] requires to store nonces and ephemeral secrets during the execution, which inflate the state. Deflation happens when the protocol is fully complete. In our perspective (and specially about instance messaging), communication is asynchronous and it can take some time before a protocol fully terminates. Hence, there is the case when several protocols run concurrently. This is the case where the state would grow with the number of incomplete sessions, just like in the instance messaging case (which we illustrate on Fig. 15).

Our Contribution

In the present work, we start with the definition of a minimal primitive called Self-Encrypted Queue (SEQ) with correctness and one-way (OW) security. It gives the minimal functionality for any PCS construction, more particularly self-encryption schemes. Then, we prove that for every SEQ primitive with states of bounded length, there is an adversary with small complexity and high probability of success to break OW security. More precisely, the probability of success is at least $\frac{1}{4n}2^{-2^{\frac{\ell+1}{\lfloor n/\Delta \rfloor}}}$ when the state size is bounded by ℓ, n is the number of active ciphertexts, and $\Delta = 1$ (or defined below). This result led us to conclude that when self-encryption is post-compromise secure, it must have a state which grows more than linearly in n.[4] This does not provide the practicality we were hoping for. Therefore, we define a refinement which is a relaxed version of post-compromise security. In layman

[4] It grows linearly if we take the key size as a memory unit. (The key size cannot have a constant bit length. Otherwise, exhaustive search breaks it with constant complexity.)

terms, we look into the following case: Maybe the first ciphertext that will be generated after an exposure is not secure, but the system could be designed to heal the security after the generation of Δ ciphertexts, where Δ is a constant parameter of our scheme. We call it Δ-PCS. We show that in refined definitions, the state size is super-linear in $\frac{n}{\Delta}$ as opposed to growing super-linear in n.

We prove that this impossibility result applies both in self-encryption and in secure messaging. In addition to this, we prove that this result is tight by constructing a simple self-encryption scheme achieving Δ-PCS with a state size matching our bounds.

After our impossibility results, we focus on few applications by borrowing already existing formal interfaces from AGJ [2] in order to add PCS security in the discussed settings. We modify the interface in a way that decryption and puncturing happens with separate function calls in case the puncturing is not always necessary. Later on, we look at secure ratcheted protocols which provides PCS security from the literature. We show that the state of these protocols grows linearly (in terms of number of keys) as they "ratchet" every time a new message is generated, hence falling into the case where $\Delta = 1$. On the other hand, we have two secure communication protocols given by Alwen, Coretti, and Dodis (called ACD and ACD-PK) [1] which model well what Signal is deploying. We observe that the state in both schemes does not grow linearly like other PCS schemes. This is due to the fact that these two protocols do not guarantee Δ-PCS for any constant Δ. In fact, healing happens only when the direction of communication changes.

We conclude that adding PCS to FS-secure systems can be succeeded at the price of a minimal state growth with proven bounds and we cannot hope for better.

Structure of the Paper

In Section 2, we define a basic PCS-secure primitive called SEQ and we prove that its state size must grow super-linearly. In Section 3, we apply this result to self-encryption. We construct a scheme based on AGJ with super-linear growth and PCS security. Finally, in Section 4, we show how to apply our result to instant secure messaging.

2 Impossibility Result

In this section, we first define a minimal primitive called Self Encrypted Queue (SEQ) achieving post-compromise security. This primitive is not meant to have any concrete application. However, we will prove that (examples of) useful primitives imply SEQ, and that SEQ must have a linearly growing state.

2.1 Definition of a Minimal Primitive

We define below a *minimal* primitive which works in two phases: It iteratively generates a sequence of plaintext/ciphertext pairs $(\mathsf{pt}, \mathsf{ct})$ by updating its state. Then, it takes the sequence of ct in the same order as generated and recovers the exact sequence of pt. The primitive is minimal in the sense that all considered applications which claim PCS must achieve this functionality and even more (such as being able to receive the list of ct in different order, or to have encryption and decryption steps mixed up). We build a limited self-encryption (actually, we build a KEM) which we call a SEQ.

▶ **Definition 1** (SEQ). *A* ***Self Encrypted Queue (SEQ)*** *is a primitive defined by*
$\mathsf{Gen}(1^\lambda) \to \mathsf{st}$ *which generates an initial state;*

Correctness at level-n:

$\mathsf{Gen}(1^\lambda) \to \mathsf{st}_0$
for $i = 1$ to n **do** ▷ fill up the queue
  $\mathsf{Enc}(\mathsf{st}_{i-1}) \to (\mathsf{st}_i, \mathsf{pt}_i, \mathsf{ct}_i)$
end for
for $i = 1$ to n **do** ▷ empty the queue
  $\mathsf{Dec}(\mathsf{st}_{n+i-1}, \mathsf{ct}_i) \to (\mathsf{st}_{n+i}, \mathsf{pt}'_i)$
  if $\mathsf{pt}_i \neq \mathsf{pt}'_i$ **then return** 0
end for
return 1

Game $\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A})$:

$\mathsf{Gen}(1^\lambda) \to \mathsf{st}_0$
for $i = 1$ to m **do**
  $\mathsf{Enc}(\mathsf{st}_{i-1}) \to (\mathsf{st}_i, \mathsf{pt}_i, \mathsf{ct}_i)$
end for
$\mathcal{A}(1^\lambda, \mathsf{st}_{m-\Delta}, \mathsf{ct}_1, \ldots, \mathsf{ct}_m) \to z$
return $1_{z=\mathsf{pt}_m}$

Figure 1 Correctness and OW games for SEQ.

$\mathsf{Enc}(\mathsf{st}) \to (\mathsf{st}', \mathsf{pt}, \mathsf{ct})$ *which updates the state and adds to the queue a new message which is* pt *in clear and* ct *in encrypted form;*

$\mathsf{Dec}(\mathsf{st}, \mathsf{ct}) \to (\mathsf{st}', \mathsf{pt}/\bot)$ *which updates the state and decrypts* ct *which leads the queue. This is deterministic.*

We say that SEQ *is* ***correct to level-n*** *if the correctness game in Fig. 1 always return 1.*[5]

The principle of this primitive is that a state is updated at every encryption/decryption so that the new state can decrypt the released ciphertext in the order they have been released. In the correctness game, the queue is filled up with $(\mathsf{ct}_1, \ldots, \mathsf{ct}_n)$, then emptied.

▶ **Definition 2.** *Let $n(\lambda)$ and $\Delta(\lambda)$ be polynomially bounded positive integer functions of a security parameter λ. We consider the $\mathsf{OW}_{m,\Delta,\lambda}$ game in Fig. 1. We say that* ***SEQ with level n is Δ-secure*** *if for any PPT adversary $\mathcal{A}$, $\lambda \mapsto \max_{1 \le m \le n} \Pr[\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A}) \to 1]$ is a negligible function.*

The value of Δ represents the time the scheme needs to heal security after an exposure. This means that Δ steps after exposing the state, the new state has become safe again and the encryptions to follow will protect confidentiality. In the game, $\mathsf{st}_{m-\Delta}$ is exposed and the goal of the adversary is to decrypt ct_m. Most secure schemes are 1-secure, because security heals after $\Delta = 1$ encryption.

It is easy to design a secure SEQ of level n with a state with $\mathcal{O}(n)$ keys inside. For instance, for any n, the scheme in Fig. 2 is a 1-secure SEQ to level n with state of size $n\lambda$, where λ is the security parameter. This SEQ is trivially correct: st accumulates all pt in a queue during encryption and releases them during decryption. It is also perfectly secure: pt is independent from the corresponding ct and from the previous states. Hence, any $\mathsf{OW}_{m,\Delta,\lambda}$ adversary has an advantage of $2^{-\lambda}$.

Ideally, states should not inflate. For that, one can count on ct to transport a helper to recover pt without having to store it in st. However, we prove next that a correct and OW-secure SEQ primitive with st in a space $\mathcal{ST}$ of size $2^{o(n \log n)}$ does not exist.

2.2 Impossibility Result

▶ **Theorem 3.** *There exists a (small) constant c such that for every probability $\alpha \in]0,1]$ and integers λ, n, ℓ, Δ, k, for every correct* SEQ *primitive of level n as in Def. 1 with* st *in a*

[5] Throughout this paper, 1_P denotes a function returning 1 if the predicate P is true, and 0 otherwise.

```
Gen(1^λ):
  st ← (λ, [])                      ▷ a list of length 0
  return st

Enc(st):
  parse st = (λ, L)
  pick pt of length λ at random
  L ← (L, pt)                       ▷ append pt in L
  st ← (λ, L)
  ct ← ⊥
  return (st, pt, ct)

Dec(st, ct):
  parse st = (λ, L)
  parse L = (pt, L')
      ▷ pt is the first length-λ element of st
  st ← (λ, L')
  return (st, pt)
```

Figure 2 A trivial SEQ.

space $\mathcal{ST}$ of size $|\mathcal{ST}| \leq 2^\ell$, there exist $m \leq n$ and an $\mathsf{OW}_{m,\Delta,\lambda}$ adversary $\mathcal{A}$ of complexity $(n-m+\Delta)T_{\mathsf{Enc}} + mT_{\mathsf{Dec}} + c$, and advantage at least

$$\Pr[\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A}) \to 1] > \frac{\alpha}{n}\left(1 - \left(\frac{1}{k} + \frac{k-1}{2}\alpha\right)^{\left\lfloor \frac{n}{\Delta} \right\rfloor} 2^\ell\right)$$

where T_{Enc} and T_{Dec} are the complexities of Enc *and* Dec.

Interestingly, for $k = 2$ and $\alpha = \frac{1}{\lfloor n/\Delta \rfloor}$, this theorem gives $\Pr[\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A}) \to 1] > \frac{\Delta}{n^2}(1 - e2^{\ell - \lfloor \frac{n}{\Delta} \rfloor})$. Thus, it is clear that $\ell \leq \lfloor \frac{n}{\Delta} \rfloor - 2$ is insecure.

We can be more precise and obtain insecurity when $\frac{\ell\Delta}{n}$ is bounded by a logarithmic term (of the security parameter). Let $\varepsilon = 2^{-\frac{\ell+1}{\lfloor n/\Delta \rfloor}}$. Th. 3 with $\alpha = \frac{\varepsilon^2}{2}$ and $k = \lceil \frac{2}{\varepsilon} \rceil$ gives the following result:

▶ **Corollary 4.** *There exists a (small) constant c such that for every integers λ, n, ℓ, and Δ, for every correct* SEQ *primitive of level n as in Def. 1 with* st *in a space $\mathcal{ST}$ of size $|\mathcal{ST}| \leq 2^\ell$, there exist $m \leq n$ and an $\mathsf{OW}_{m,\Delta,\lambda}$ adversary $\mathcal{A}$ of complexity $(n-m+\Delta)T_{\mathsf{Enc}} + mT_{\mathsf{Dec}} + c$, and advantage at least*

$$\Pr[\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A}) \to 1] > \frac{1}{4n} 2^{-2\frac{\ell+1}{\lfloor n/\Delta \rfloor}}$$

where T_{Enc} and T_{Dec} are the complexities of Enc *and* Dec.

This means that the state needs a size $\boldsymbol{\ell}$ such that

$$\boldsymbol{\ell > \frac{1}{2}\left\lfloor \frac{n}{\Delta} \right\rfloor \log_2 \frac{1}{4n\varepsilon} - 1} \tag{1}$$

to achieve $\boldsymbol{\Delta}$-security up to $\boldsymbol{n}$ encryptions with advantage bounded by $\boldsymbol{\varepsilon}$. For $\varepsilon = 2^{-\lambda}$ and $n = \mathsf{Poly}(\lambda)$, the dominant term is $\frac{\lambda n}{2\Delta}$.

We can now prove Th. 3:

Proof. Let us consider a correct primitive of level n with st in a space $\mathcal{ST}$ such that $|\mathcal{ST}| \leq 2^\ell$. We will show that it is insecure. To do so, we will first express that the state st after n encryptions are *constrained.* Namely, constraints are that st must decrypt the generated sequence of ct correctly. The constraints increase with n, and the set of possible st values which make decryption correct decreases. The set of constrained states does *not* decrease exponentially because of the surprising existence of "super states" which are able to decrypt more than their constraints. Namely, super states can decrypt universaly, including encryptions from the “future” which have not been generated yet. This is counter-intuitive. This set of super states is a hard core in the set of constrained states. We show that the set of constrained which are non-super states *does* decrease exponentially. Hence, by taking n large enough, constrained states become all super states: the state after n encryptions must be a super state. We use the property of the super state to mount an attack.

We first define notations. We extend the Enc and Dec functions. First of all, with random coins ρ, we write $\mathsf{Enc}(\mathsf{st};\rho) = (\mathsf{st}', \mathsf{pt}, \mathsf{ct})$ and consider Enc as deterministic with explicit coins. For $X \in \{\mathsf{Enc}, \mathsf{Dec}\}$ and $y \in \{\mathsf{st}, \mathsf{pt}, \mathsf{ct}\}$, we denote by $X_{\mathsf{o\_}y}$ the generated **o**utput of type y by the X operation: for both Enc and Dec, the output components define subfunctions $\mathsf{Enc}_{\mathsf{o\_st}}$, $\mathsf{Enc}_{\mathsf{o\_pt}}$, $\mathsf{Enc}_{\mathsf{o\_ct}}$, $\mathsf{Dec}_{\mathsf{o\_st}}$, $\mathsf{Dec}_{\mathsf{o\_pt}}$ by

$$\begin{aligned}\mathsf{Enc}(\mathsf{st};\rho) &= (\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st};\rho), \mathsf{Enc}_{\mathsf{o\_pt}}(\mathsf{st};\rho), \mathsf{Enc}_{\mathsf{o\_ct}}(\mathsf{st};\rho))\\ \mathsf{Dec}(\mathsf{st},\mathsf{ct}) &= (\mathsf{Dec}_{\mathsf{o\_st}}(\mathsf{st},\mathsf{ct}), \mathsf{Dec}_{\mathsf{o\_pt}}(\mathsf{st},\mathsf{ct}))\end{aligned}$$

We further extend those functions with a variable number of inputs ρ or ct. We define

$$\begin{aligned}\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st},\rho_1,\ldots,\rho_i) &= \mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st},\rho_1,\ldots,\rho_{i-1});\rho_i)\\ \mathsf{Dec}_{\mathsf{o\_st}}(\mathsf{st},\mathsf{ct}_1,\ldots,\mathsf{ct}_i) &= \mathsf{Dec}_{\mathsf{o\_st}}(\mathsf{Dec}_{\mathsf{o\_st}}(\mathsf{st},\mathsf{ct}_1,\ldots,\mathsf{ct}_{i-1}),\mathsf{ct}_i)\end{aligned}$$

with the convention that $\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st}) = \mathsf{st}$ and $\mathsf{Dec}_{\mathsf{o\_st}}(\mathsf{st}) = \mathsf{st}$, i.e., the functions with zero coins do nothing but returning st unchanged. Next, $\mathsf{Enc}_{\mathsf{o\_pt}}(\mathsf{st},\rho_1,\ldots,\rho_i)$ is the list of generated pt, $\mathsf{Enc}_{\mathsf{o\_ct}}(\mathsf{st},\rho_1,\ldots,\rho_i)$ is the list of generated ct, and $\mathsf{Dec}_{\mathsf{o\_pt}}(\mathsf{st},\mathsf{ct}_1,\ldots,\mathsf{ct}_i)$ is the list of decrypted pt:

$$\begin{aligned}\mathsf{Enc}_{\mathsf{o\_pt}}(\mathsf{st},\rho_1,\ldots,\rho_i) &= (\mathsf{Enc}_{\mathsf{o\_pt}}(\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st},\rho_1,\ldots,\rho_{j-1});\rho_j))_{j=1,\ldots,i}\\ \mathsf{Enc}_{\mathsf{o\_ct}}(\mathsf{st},\rho_1,\ldots,\rho_i) &= (\mathsf{Enc}_{\mathsf{o\_ct}}(\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st},\rho_1,\ldots,\rho_{j-1});\rho_j))_{j=1,\ldots,i}\\ \mathsf{Dec}_{\mathsf{o\_pt}}(\mathsf{st},\mathsf{ct}_1,\ldots,\mathsf{ct}_i) &= (\mathsf{Dec}_{\mathsf{o\_pt}}(\mathsf{Dec}_{\mathsf{o\_st}}(\mathsf{st},\mathsf{ct}_1,\ldots,\mathsf{ct}_{j-1}),\mathsf{ct}_j))_{j=1,\ldots,i}\end{aligned}$$

Let st_n be the state which is obtained after n encryptions, before starting the decryption phase. In order to characterize the constraints on st_n coming from the first i encryptions, we introduce a set $C[r_i]$ corresponding to (and indexed with) each update operation $r_i = (\mathsf{st}_0, \rho_1, \ldots, \rho_i)$. Due to correctness, st_n must decrypt $\mathsf{Enc}_{\mathsf{o\_ct}}(r_i)$ to $\mathsf{Enc}_{\mathsf{o\_pt}}(r_i)$. Hence, we define the set of states which are constrained to r_i by

$$C[r_i] = \{\mathsf{st} \in \mathcal{ST}; \mathsf{Dec}_{\mathsf{o\_pt}}(\mathsf{st}, \mathsf{Enc}_{\mathsf{o\_ct}}(r_i)) = \mathsf{Enc}_{\mathsf{o\_pt}}(r_i)\}$$

Clearly, for any i and any $\mathsf{st}_0, \rho_1, \ldots, \rho_n$, we have

$$\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st}_0, \rho_1, \ldots, \rho_n) \in C[\mathsf{st}_0, \rho_1, \ldots, \rho_i]$$

We note that $C[r_0]$, where $r_0 = \mathsf{st}_0$ is the set of states subject to no restriction, hence $C[\mathsf{st}_0] = \mathcal{ST}$. Furthermore, we note that

$$C[r_n] \subseteq \cdots \subseteq C[r_2] \subseteq C[r_1] \subseteq C[r_0] = \mathcal{ST}$$

A state in $C[r_{i-\Delta}]$ decrypts well the first $i - \Delta$ ciphertexts. It may also be element of $C[r_{i-\Delta}, \rho_{i-\Delta+1}, \ldots, \rho_i]$ if it decrypts the next Δ ciphertexts which are produced with coins $\rho_{i-\Delta+1}, \ldots, \rho_i$. It may also be in $C[r_{i-\Delta}, \rho'_{i-\Delta+1}, \ldots, \rho'_i]$ and decrypt Δ ciphertexts produced with other coins. With good probability, some state may actually have the “super-power” to decrypt ciphertexts produced with Δ more random coins. We call those states the *super states*. Intuitively, this is unexpected to happen but we show below that super-states exist and an adversary can build some easily.

More concretely, let $\alpha > 0$ be the probability from the statement of the theorem. We define a set of super states for $r_{j-\Delta} = (\mathsf{st}_0, \rho_1, \ldots, \rho_{j-\Delta})$:

$$S[r_{j-\Delta}] = \left\{ \mathsf{st} \in \mathcal{ST};\ \Pr_{\rho'_{j-\Delta+1},\ldots,\rho'_j}[\mathsf{st} \in C[r_{j-\Delta}, \rho'_{j-\Delta+1}, \ldots, \rho'_j]] > \alpha \right\}$$

This set $S[r_{j-\Delta}]$ defines a set of states which are α-likely to decrypt a "fork" in the sequence of random coins. (See Fig. 3.)

We note that $S[r_{j-\Delta}] \subseteq C[r_{j-\Delta}]$ since for $\mathsf{st} \in S[r_{j-\Delta}]$, there must exist (due to a non-zero probability) $\rho'_{j-\Delta+1}, \ldots, \rho'_j$ such that

$$\mathsf{st} \in C[r_{j-\Delta}, \rho'_{j-\Delta+1}, \ldots, \rho'_j] \subseteq C[r_{j-\Delta}]$$

We define a union of super states as follows:

$$S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{n-\Delta}] = S[\mathsf{st}_0] \cup S[\mathsf{st}_0; \rho_1] \cup \cdots \cup S[\mathsf{st}_0; \rho_1, \ldots, \rho_{n-\Delta}]$$

Clearly

$$S^{\cup}[r_{n-\Delta}] \supseteq \cdots \supseteq S^{\cup}[r_1] \supseteq S^{\cup}[r_0]$$

The idea of the proof is to show that states with too many constraints tend to become super-states. Namely, we first prove that for n large enough, $C[r_n]$ is included in $S^{\cup}[r_{n-\Delta}]$ with large probability p. This means that after n encryptions, a state becomes a super-state. Hence, this state belongs to some $S[r_{m-\Delta}]$, with a random $m \le n$. We now take a fixed value of m which is taken with probability at least $\frac{1}{n}$. (It exists, due to the pigeon-hole principle.) We take n encryptions from random coins $\mathsf{st}_0, \rho_1, \ldots, \rho_{m-\Delta}, \rho'_{m-\Delta+1}, \ldots, \rho'_n$. We deduce that there is a probability at least $\frac{p}{n}$ to get a state st'_n in $S[r_{m-\Delta}]$. If it happens, st'_n decrypts what is generated by the fork $\mathsf{st}_0, \rho_1, \ldots, \rho_m$ with probability at least α (by definition of the super states). We define an adversary that exploits this fact in Fig. 3. The m encryptions with $\mathsf{st}_0, \rho_1, \ldots, \rho_m$ are generated by the game, the state $\mathsf{st}_{m-\Delta}$ leaks, and the adversary can fork to construct st'_n from it. We obtain the success probability of the adversary in the $\mathsf{OW}_{m,\Delta,\lambda}$ game:

$$\Pr[\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A}) \to 1] > \frac{\alpha p}{n} \tag{2}$$

In what follows, we show that $p \ge 1 - \left(\frac{1}{k} + \frac{k-1}{2}\alpha\right)^{\left\lfloor \frac{n}{\Delta} \right\rfloor} 2^{\ell}$.

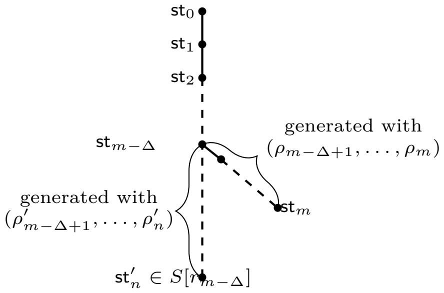
Let i be an integer. We consider for the moment that $\mathsf{st}_0, \rho_1, \ldots, \rho_{i-\Delta}$ are fixed. For simplicity, we denote

$$\begin{array}{rcl} S^{\cup}_{i-\Delta} & = & S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{i-2\Delta}] \\ S^{\cup}_i & = & S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{i-\Delta}] \end{array}$$

$$\begin{array}{rcl} C_{i-\Delta} & = & C[\mathsf{st}_0, \rho_1, \ldots, \rho_{i-\Delta}] \\ C_i(\vec{\rho}) & = & C[\mathsf{st}_0, \rho_1, \ldots, \rho_{i-\Delta}, \vec{\rho}] \end{array}$$

for a vector $\vec{\rho}$ of dimension Δ. We take k independent random Δ-dimensional vectors $\vec{\rho}_j$, for integers $j = 1, \ldots, k$ and we define $C_{i,j} = C_i(\vec{\rho}_j)$. ($k$ is defined in the statement of the Lemma.) Given $\vec{\rho}_j$ fixed and some $\mathsf{st} \in C_{i,j} - S^{\cup}_i$ fixed, we have $\mathsf{st} \notin S^{\cup}_i$ meaning that $\mathsf{st} \notin S[\mathsf{st}_0, \rho_1, \ldots, \rho_{i-\Delta}]$, thus

$$\Pr_{\vec{\rho}_{j'}}[\mathsf{st} \in C_{i,j'}] \le \alpha$$



$\underline{\mathcal{A}(1^\lambda, \mathsf{st}_{m-\Delta}, \mathsf{ct}_1, \ldots, \mathsf{ct}_m)\text{:}}$

$\mathsf{st}'_{m-\Delta} \leftarrow \mathsf{st}_{m-\Delta}$
for $i = m - \Delta + 1$ to n **do**
  pick ρ'_i
  $\mathsf{Enc}(\mathsf{st}'_{i-1}; \rho'_i) \to (\mathsf{st}'_i, \mathsf{pt}'_i, \mathsf{ct}'_i)$
end for
for $i = 0$ to $m - 1$ **do**
  $\mathsf{Dec}(\mathsf{st}'_{n+i}, \mathsf{ct}_{i+1}) \to (\mathsf{st}'_{n+i+1}, \overline{\mathsf{pt}}_{i+1})$
end for
return $\overline{\mathsf{pt}}_m$

Figure 3 Starting from state st_0 and applying m encryption that generates $\mathsf{ct}_1, \ldots, \mathsf{ct}_m$, we hope that leaking st_m and forking to n encryptions in total will end up in $\mathsf{st}'_n \in S[r_{m-\Delta}]$. Therefore, st'_n decrypts all the ciphertext with probability at least α.

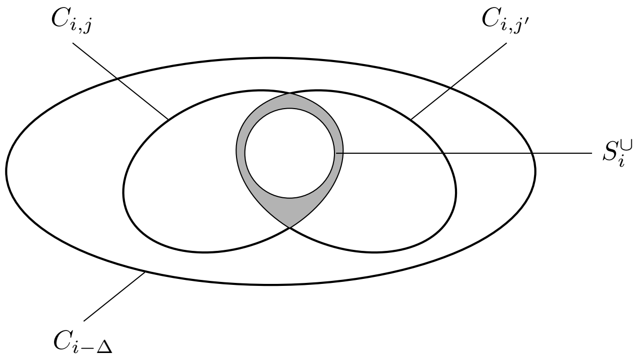


Figure 4 Illustration of the intersection $(C_{i,j'} - S_i^\cup) \cap (C_{i,j} - S_i^\cup)$.

for any $\vec{\rho}_{j'}$ independent vector indexed with $j' \neq j$, by definition of S_i and $C_{i,j'}$. We count

$$|(C_{i,j'} - S_i^\cup) \cap (C_{i,j} - S_i^\cup)| = \sum_{\mathsf{st} \in C_{i,j} - S_i^\cup} 1_{\mathsf{st} \in C_{i,j'}}$$

We obtain

$$\mathop{\mathsf{E}}_{\vec{\rho}_{j'}} [|(C_{i,j'} - S_i^\cup) \cap (C_{i,j} - S_i^\cup)|] \leq \alpha |C_{i,j} - S_i^\cup| \leq \alpha |C_{i-\Delta} - S_{i-\Delta}^\cup|$$

for any j, j', and $\vec{\rho}_j$ with $j \neq j'$. This is illustrated in Fig. 4. Clearly, we can then randomize $\vec{\rho}_j$ and obtain

$$\mathsf{E}\,[|(C_{i,j'} - S_i^\cup) \cap (C_{i,j} - S_i^\cup)|] \leq \alpha |C_{i-\Delta} - S_{i-\Delta}^\cup|$$

for any j and j' with $j \neq j'$.

Let $A_j = C_{i,j} - S_i^\cup$. This denotes one of the k subsets of $A = C_{i-\Delta} - S_{i-\Delta}^\cup$. We have

$$\sum_{j=1}^{k} |A_j| \leq |A| + \sum_{1 \leq j < j' \leq k} |A_j \cap A_{j'}|$$

Indeed, any element x of A occurring in exactly m subsets A_j is counted m times on the left-hand side and $1 + \frac{m(m-1)}{2}$ times on the right-hand side. However, $m \leq 1 + \frac{m(m-1)}{2}$ for every integer m. We deduce

$$\mathsf{E}\left[\sum_{j=1}^{k}|C_{i,j} - S_i^{\cup}|\right] \leq \left(1 + \frac{k(k-1)}{2}\alpha\right)|C_{i-\Delta} - S_{i-\Delta}^{\cup}|$$

Given that all $\mathsf{E}\left[|C_{i,j} - S_i^{\cup}|\right]$ are equal, we have proven that

$$\mathsf{E}_{\vec{\rho}}\left[|C_i(\vec{\rho}) - S_i^{\cup}|\right] \leq \left(\frac{1}{k} + \frac{k-1}{2}\alpha\right)|C_{i-\Delta} - S_{i-\Delta}^{\cup}|$$

We can now randomize $\rho_1, \ldots, \rho_{n-\Delta}$ as well and obtain

$$\mathsf{E}\left[|C[\mathsf{st}_0, \rho_1, \ldots, \rho_n] - S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{n-\Delta}]|\right] \leq \left(\frac{1}{k} + \frac{k-1}{2}\alpha\right)^{\lfloor \frac{n}{\Delta} \rfloor}|\mathcal{ST}|$$

We bound $|\mathcal{ST}| \leq 2^\ell$ and $\Pr[E \neq \emptyset] \leq \mathsf{E}[|E|]$ (due to the Markov inequality) for a random set E and obtain

$$\Pr[C[\mathsf{st}_0, \rho_1, \ldots, \rho_n] - S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{n-\Delta}] \neq \emptyset] \leq \left(\frac{1}{k} + \frac{k-1}{2}\alpha\right)^{\lfloor \frac{n}{\Delta} \rfloor} 2^\ell$$

By assumption on the size of $\mathcal{ST}$, for n large enough, we obtain that the set difference $C[\mathsf{st}_0, \rho_1, \ldots, \rho_n] - S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{n-\Delta}]$ is likely to be empty which means that the states in $C[\mathsf{st}_0, \rho_1, \ldots, \rho_n]$ are super states. By the definition of $C[r_n]$, $\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st}_0; \rho_1, \ldots, \rho_n) \in C[\mathsf{st}_0; \rho_1, \ldots, \rho_n]$. Hence, $\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st}_0; \rho_1, \ldots, \rho_n)$ is likely to be in $S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{n-\Delta}]$. More precisely,

$$\Pr[\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st}_0, \rho_1, \ldots, \rho_n) \notin S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{n-\Delta}]] \leq \left(\frac{1}{k} + \frac{k-1}{2}\alpha\right)^{\lfloor \frac{n}{\Delta} \rfloor} 2^\ell$$

If $\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st}_0, \rho_1, \ldots, \rho_n) \in S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{n-\Delta}]$, it means there exists (at least) one $m \leq n$ such that

$$\Pr_{\vec{\rho}'}[\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st}_0, \rho_1, \ldots, \rho_n) \in C(\mathsf{st}_0, \rho_1, \ldots, \rho_{m-\Delta}, \vec{\rho}')] > \alpha$$

Therefore, we obtain the success probability in the $\mathsf{OW}_{m,\Delta,\lambda}$ game (from Eq. (2)):

$$\Pr[\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A}) \to 1] > \frac{\alpha}{n}\left(1 - \left(\frac{1}{k} + \frac{k-1}{2}\alpha\right)^{\lfloor \frac{n}{\Delta} \rfloor} 2^\ell\right)$$

The complexity of $\mathcal{A}$ is $n - m + \Delta$ encryptions and m decryptions. ◀

Uniform Impossibility Result

Our Th. 3 and Cor. 4 are *non-uniform* in the sense that the parameter m depends on λ in an unknown manner. However, $\mathcal{A}$ is constructed in a polynomially bounded manner based on m. Thus, by guessing m, we obtain a uniform result with advantage divided by n.

3 Self-Ratchet

3.1 Definitions

Consider a self-ratcheted scheme $\mathsf{SR} = (\mathsf{lg}, \mathsf{Init}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Punc})$ with the following syntax:

- $\mathsf{lg}(\lambda)$ (length of the plaintext)
- $\mathsf{SR.Init}(1^\lambda) \xrightarrow{\$} \mathsf{st}$ (output an initial state for the device)
- $\mathsf{SR.Enc}(\mathsf{st}, \mathsf{pt}) \xrightarrow{\$} (\mathsf{st}', \mathsf{ct})$ (update the state while encrypting $\mathsf{pt} \in \{0,1\}^{\mathsf{lg}(\lambda)}$)
- $\mathsf{SR.Dec}(\mathsf{st}, \mathsf{ct}) \to \mathsf{pt}$ or $\perp$ (decrypt ct into pt)
- $\mathsf{SR.Punc}(\mathsf{st}, \mathsf{ct}) \to \mathsf{st}'$ (update the state by puncturing ct in st)

In our settings, there exists a device following a protocol which produces some pt/ct for itself so that it can eventually decrypt ct to recover pt in the future. Encryption is stateful. The protocol makes sure that when the device should no longer be able to decrypt ct and should be secure against any future state exposure, it can "puncture" the state. This means that the state st which can decrypt ct is replaced by a new (punctured) state st' so that ct is not decryptable by st'. With this notion, we aim at forward security and PCS.

▶ **Definition 5** (SR). *A **self-ratcheted scheme (SR) of level n** is a primitive* $\mathsf{SR} = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Punc})$ *which is **n-correct** in the sense that for any sequence* sched*, the game in Fig. 5 never returns 1. Here,* sched *is a sequence of scheduled instructions which can be of three different types:* ("Enc", pt) *(encrypt plaintext* pt*),* ("Dec", j) *(decrypt the j-th produced ciphertext), and* ("Punc", j) *(puncture the j-th produced ciphertext).*

The correctness notion must consider any order of Enc/Dec/Punc instructions. This is what sched is modeling. We describe what should happen when this sequence of instructions is sched. Actually, we declare in L_{ct} the ciphertexts which are "active" and we put in L_{pt} how they are expected to decrypt.

This definition assumes that the number of "active" ciphertexts remains bounded by a parameter n (line number 5).

Compared to SEQ, an SR does not update the state during decryption (this is rather done by a separate function) and decryption can be done in any order of the ciphertexts (i.e., not only in the oder they have been created). As applications will show, SR appears to be a most wanted primitive.

Application to Cloud Storage

SR schemes can be used for cloud storage where a client wants to store her files on the cloud in an encrypted form. Ideally, a single file is encrypted with SR.Enc to obtain a ct. For retrieval, the SR.Dec is run to decrypt the file. Eventually, when the client wants to remove the file from the cloud, the protocol will puncture her state for ct. The first desired security is that after a client erases an encrypted file, even though a copy was illegally kept and the state of the client later leaks, the file is unrecoverable. This is forward security. With SR, it is achieved by puncturing. The second desired security is that after the state of a client has leaked, if the client wants to store a new file in the cloud, this file should be safe, as long as no exposure occurs during the activity time of this file. This is post-compromise security. It is achieved by what we call self-ratchet.

One problem specific to cloud storage is that files are typically big and SR should handle them in encryption, decryption, and puncturing. One common approach is to use a domain expander based on a hybrid construction. Like the KEM/DEM hybrid cryptosystems, we can use SR to encrypt an ephemeral key K and symmetrically encrypt the plaintext with K.

We could also add key rotation, if required, by using SR to encrypt the encryption key: to encrypt a file pt, we pick a random key k (in the key domain of the key rotation scheme) and we run $\mathsf{ct}_1 \leftarrow \mathsf{SR.Enc}(\mathsf{st}, k)$. Then, we encrypt pt with k following the key rotation scheme and obtain a header ct_2 and a ciphertext ct_3. The final ciphertext is $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3)$. To

$\mathsf{SR.Init}(1^\lambda) \xrightarrow{\$} \mathsf{st}$
set lists L_{pt} and L_{ct} to empty
for $i = 1$ to $|\mathsf{sched}|$ **do**
  if sched_i parses as ("Enc", pt) for some pt **then**
    if the number of L_{ct} entries which are different from $\perp$ is at most $n-1$ **then**
      $\mathsf{SR.Enc}(\mathsf{st}, \mathsf{pt}) \to (\mathsf{st}, \mathsf{ct})$
      $L_{\mathsf{pt}} \leftarrow (L_{\mathsf{pt}}, \mathsf{pt})$
      $L_{\mathsf{ct}} \leftarrow (L_{\mathsf{ct}}, \mathsf{ct})$
    end if
  else if sched_i parses as ("Dec", j) for some j **then**
    if $L_{\mathsf{ct}}[j]$ exists and $L_{\mathsf{ct}}[j] \neq \perp$ **then**
      $\mathsf{SR.Dec}(\mathsf{st}, L_{\mathsf{ct}}[j]) \to \mathsf{pt}$
      if $\mathsf{pt} \neq L_{\mathsf{pt}}[j]$ **then return** 1
    end if
  else if sched_i parses as ("Punc", j) for some j **then**
    if $L_{\mathsf{ct}}[j]$ exists and $L_{\mathsf{ct}}[j] \neq \perp$ **then**
      $\mathsf{SR.Punc}(\mathsf{st}, L_{\mathsf{ct}}[j]) \to \mathsf{st}$
      $L_{\mathsf{ct}}[j] \leftarrow \perp$
    end if
  end if
end for
return 0

Figure 5 Correctness game for SR of level n.

rotate the key k, we puncture st with ct_1, run the key rotation scheme on $(\mathsf{ct}_2, \mathsf{ct}_3)$ to get a new key k' and new $(\mathsf{ct}'_2, \mathsf{ct}'_3)$, and run $\mathsf{ct}'_1 \leftarrow \mathsf{SR.Enc}(\mathsf{st}, k')$ to form $\mathsf{ct}' = (\mathsf{ct}'_1, \mathsf{ct}'_2, \mathsf{ct}'_3)$.

Application to 0-RTT Session Resumption

SR schemes can be used for 0-RTT session resumption. Essentially, a server having a secure connection with a client using a key K would use $\mathsf{SR.Enc}(\mathsf{st}, \mathsf{K})$ to issue a ticket ct and send ct to the client. To resume a session, the client, who kept K and ct, would resend ct to the server who would use SR.Dec to recover K. The server might also immediately puncture it to avoid any replay of the ticket ct and for forward security.

Previous Work on 0-RTT Session Resumption

Def. 5 is more general than the definition of 0-RTT session resumption [2]. The differences are as follows:

- the notations for 0-RTT session resumption are Setup, TicketGen, and ServerRes instead of Init, Enc, Dec;
- SR separates SR.Dec and SR.Punc instead of having both functionalities in ServerRes.

There is no formal definition of correctness for 0-RTT session resumption in Aviram et al. [2]. However, we can fairly assume it is the same as our notion of correctness in Def. 5, but when sequences sched are limited such that every decryption is followed by puncturing: for all i and j, if $\mathsf{sched}_i = (\text{"Dec"}, j)$ then $\mathsf{sched}_{i+1} = (\text{"Punc"}, j)$. In 0-RTT session resumption, it makes sense to merge SR.Dec with SR.Punc as one of the security goal is precisely to prevent a ct to be replayed. For cloud storage, the client may need to decrypt the same ct several times before she removes the file from the cloud. Hence, we keep SR.Dec and SR.Punc separate.

We adapt the security definition of 0-RTT session resumption with our notations to which we add specific instructions for post-compromise security. We define the $\mathsf{IND}^{\mathsf{SR},\mathsf{opt}}_{b,n,\Delta,\lambda}(\mathcal{A})$ game in Fig. 6. We also generalize it to adaptive security. In the AGJ security model, the game starts with many OEnc and only after that, the adversary can play with oracles except OEnc (it is somehow non-adaptive). The AGJ model uses $\mathsf{opt} = \{\mathsf{noPCS}, \mathsf{replay}\}$ and it is formalized for key establishment rather than encryption. (This means that there is a Test oracle to test a decryption instead of a $\mathsf{Challenge}$ oracle to get an encryption challenge.)

▶ **Definition 6** (SR security). *Let $n(\lambda)$ and $\Delta(\lambda)$ be polynomially bounded positive integer functions of a security parameter λ. The option set* opt *specifies some variants in the game in Fig. 6. The advantage is*

$$\mathsf{Adv}_{n,\Delta,\lambda}^{\mathsf{IND}^{\mathsf{SR},\mathsf{opt}}}(\mathcal{A}) = \left|\Pr\left[\mathsf{IND}_{1,n,\Delta,\lambda}^{\mathsf{SR},\mathsf{opt}}(\mathcal{A}) \to 1\right] - \Pr\left[\mathsf{IND}_{0,n,\Delta,\lambda}^{\mathsf{SR},\mathsf{opt}}(\mathcal{A}) \to 1\right]\right|$$

We say that SR ***is*** IND-opt ***secure at level n with delay Δ*** *if for any PPT adversary $\mathcal{A}$, $\lambda \mapsto \mathsf{Adv}_{n,\Delta,\lambda}^{\mathsf{IND}^{\mathsf{SR},\mathsf{opt}}}(\mathcal{A})$ is a negligible function.*

When "replay" $\in \mathsf{opt}$, the security notion aims to address replay attacks. It enforces puncturing after decryption. Hence, decryption must puncture, as well. When "noPCS" $\in \mathsf{opt}$, the security notion aims to capture forward security *without* post-compromise security. Absence of noPCS in opt is a stronger security notion as it captures FS and PCS together.

Game $\mathsf{IND}_{b,n,\Delta,\lambda}^{\mathsf{SR},\mathsf{opt}}(\mathcal{A})$:

$\mathsf{Init}(1^\lambda) \to \mathsf{st}$
$\mathsf{Active}, \mathsf{Revealed} \leftarrow \emptyset$
$\mathsf{challenged} \leftarrow \mathsf{false}$
$\mathsf{AfterExp} \leftarrow \Delta$
$\mathcal{A}^{\mathsf{OEnc},\mathsf{ODec},\mathsf{Challenge},\mathsf{OPunc},\mathsf{OExp}}(1^\lambda) \to b^*$
return b^*

Oracle $\mathsf{OEnc}(\mathsf{pt})$:

if $|\mathsf{Active}| \geq n$ **then return** $\perp$
$\mathsf{SR.Enc}(\mathsf{st}, \mathsf{pt}) \to (\mathsf{st}, \mathsf{ct})$
$\mathsf{Active} \leftarrow \mathsf{Active} \cup \{\mathsf{ct}\}$
$\mathsf{Revealed} \leftarrow \mathsf{Revealed} \cup \{\mathsf{ct}\}$
$\mathsf{AfterExp} \leftarrow \mathsf{AfterExp} + 1$
return ct

Oracle $\mathsf{ODec}(\mathsf{ct})$:

if $\mathsf{ct} \in \mathsf{Active} - \mathsf{Revealed}$ **then**
  return $\perp$
end if
$\mathsf{SR.Dec}(\mathsf{st}, \mathsf{ct}) \to r$
if "replay" $\in \mathsf{opt}$ **then** $\mathsf{OPunc}(\mathsf{ct})$
return r

Oracle $\mathsf{OPunc}(\mathsf{ct})$:

$\mathsf{SR.Punc}(\mathsf{st}, \mathsf{ct}) \to \mathsf{st}$
$\mathsf{Active} \leftarrow \mathsf{Active} - \{\mathsf{ct}\}$
$\mathsf{Revealed} \leftarrow \mathsf{Revealed} - \{\mathsf{ct}\}$
return

Oracle $\mathsf{Challenge}(\mathsf{pt}_1)$:

if $\mathsf{challenged}$ **then return** $\perp$
if $|\mathsf{Active}| \geq n$ **then return** $\perp$
if $\mathsf{AfterExp} < \Delta$ **then return** $\perp$
pick pt_0 of same length as pt_1 at random
$\mathsf{SR.Enc}(\mathsf{st}, \mathsf{pt}_b) \to (\mathsf{st}, \mathsf{ct})$
$\mathsf{Active} \leftarrow \mathsf{Active} \cup \{\mathsf{ct}\}$
$\mathsf{AfterExp} \leftarrow \mathsf{AfterExp} + 1$
$\mathsf{challenged} \leftarrow \mathsf{true}$
return ct

Oracle $\mathsf{OExp}()$:

if ($\neg\mathsf{challenged}$ and "noPCS" $\in \mathsf{opt}$) or ($\mathsf{Active} - \mathsf{Revealed} \neq \emptyset$) **then**
  return $\perp$
end if
$\mathsf{AfterExp} \leftarrow 0$
return st

Figure 6 Indistinguishability game for self-ratchet.

$\underline{S.\mathsf{Gen}} = \mathsf{SR.Init}$

$\underline{S.\mathsf{Enc}(\mathsf{st})}$:

pick $\mathsf{K} \in \{0,1\}^{\mathsf{lg}(\lambda)}$ at random
$\mathsf{SR.Enc}(\mathsf{st}, \mathsf{K}) \to (\mathsf{st}', \mathsf{ct})$
return $(\mathsf{st}', \mathsf{K}, \mathsf{ct})$

$\underline{S.\mathsf{Dec}(\mathsf{st}, \mathsf{ct})}$:

$\mathsf{SR.Dec}(\mathsf{st}, \mathsf{ct}) \to \mathsf{K}$
if $\mathsf{K} \neq \perp$ **then** $\mathsf{SR.Punc}(\mathsf{st}, \mathsf{ct}) \to \mathsf{st}$
return $(\mathsf{st}, \mathsf{K})$

Figure 7 SEQ from SR.

$\underline{\mathcal{A}^{\mathsf{OEnc},\mathsf{ODec},\mathsf{Challenge},\mathsf{OPunc},\mathsf{OExp}}(1^\lambda)}$:

pick $m \in \{\Delta, \ldots, n\}$
for $i = 1$ to $m - \Delta$ **do**
  pick pt_i at random
  $\mathsf{OEnc}(\mathsf{pt}_i) \to ct_i$
end for
$\mathsf{OExp}() \to st_{m-\Delta}$
for $i = m - \Delta + 1$ to $m - 1$ **do**
  pick pt_i at random
  $\mathsf{OEnc}(\mathsf{pt}_i) \to ct_i$
end for
pick pt_m at random
$\mathsf{Challenge}(\mathsf{pt}_m) \to \mathsf{ct}_m$
$\mathcal{B}(1^\lambda, \mathsf{st}_{m-\Delta}, \mathsf{ct}_1, \ldots, \mathsf{ct}_m) \to z$
return $1_{z=\mathsf{pt}_m}$

Figure 8 Adversary against SR based on an adversary for SEQ.

3.2 Impossibility Result

▶ **Theorem 7.** *For every integer n, ℓ, $\Delta > 0$ and any n-correct self-ratcheted scheme* SR *following Def. 5, and such that* st *belongs to a space of size bounded by 2^ℓ, there exist a (small) constant c and an adversary of complexity bounded by $(n + \Delta)(T_{\mathsf{Enc}} + T_{\mathsf{Dec}} + T_{\mathsf{Punc}} + 1) + c$ having advantage*

$$\mathsf{Adv}_{n,\Delta,\lambda}^{\mathsf{IND}^{\mathsf{SR},\mathsf{opt}}}(\mathcal{A}) > \frac{1}{4n^2} 2^{-2^{\frac{\ell+1}{\lfloor n/\Delta \rfloor}}} - 2^{-\mathsf{lg}(\lambda)}$$

for $\mathsf{opt} = \perp$ *and* $\mathsf{opt} = \mathsf{replay}$*, and where $T_\$$ is the complexity to pick an element of $\{0,1\}^{\mathsf{lg}(\lambda)}$ at random and T_{Enc}, T_{Dec} and T_{Punc} are the complexities of* Enc, Dec *and* Punc.

Proof. We construct a SEQ from a self-ratcheted protocol SR in Fig. 7. Clearly, the n-correctness of SR implies the n-correctness of S for any n. The SEQ scheme only imposes ciphertexts to be received in the same order as they have been produced.

Due to Cor. 4, there exists m and an $\mathsf{OW}_{m,\Delta,\lambda}$ adversary $\mathcal{B}$ such that $\Pr[\mathsf{OW}_{m,\Delta,\lambda} \to 1] = p$ with $p > \frac{1}{4n} 2^{-2^{\frac{\ell+1}{\lfloor n/\Delta \rfloor}}}$. $\mathcal{B}$ is constructed uniformly from m. Then, we can construct an $\mathsf{IND}_{b,n,\Delta,\lambda}^{\mathsf{SR},\mathsf{opt}}$ adversary $\mathcal{A}$ who guesses m as in Fig. 8.

The Challenge oracle encrypts pt_m which is either pt_m or random. Since $\mathcal{A}$ simulates well the $\mathsf{OW}_{m,\Delta,\lambda}$ game, we have $\Pr[z = \mathsf{pt}_m] \geq \frac{p}{n}$. Hence, $\Pr[\mathsf{IND}_{1,n,\Delta}^{\mathsf{SR},\mathsf{opt}} \to 1] = \frac{p}{n}$ and $\Pr[\mathsf{IND}_{0,n,\Delta}^{\mathsf{SR},\mathsf{opt}} \to 1] = 2^{-\mathsf{lg}(\lambda)}$. Hence, the advantage is $\frac{p}{n} - 2^{-\mathsf{lg}(\lambda)}$.

The adversary $\mathcal{A}$ picks m plaintexts and issues $m - 1$ OEnc queries, one OExp query and one Challenge query, and then simulates an $\mathsf{OW}_{m,\Delta,\lambda}$ adversary $\mathcal{B}$. The complexity of $\mathcal{B}$ is the complexity of $n - m + \Delta$ encryptions and m decryptions, and the complexities of $S.\mathsf{Enc}$ and $S.\mathsf{Dec}$ are respectively $T_{\mathsf{Enc}} + T_\$$ and $T_{\mathsf{Dec}} + T_{\mathsf{Punc}}$. The complexity of $\mathcal{A}$ therefore is $n - m + \Delta$ encryptions, m decryptions, m punctuations, $m + 1$ oracle calls and $(n + \Delta)$ random selections. ◀

```
SR.Init(1^λ):
  st ← (0, [])
                          ▷ a counter set to 0 and an empty list
  return st

SR.Dec(st, ct):
  parse st = (c, L) and ct = (i, ct_0)
  FSSR.Dec(L[i], ct_0) → pt
  return pt

SR.Punc(st, ct):
  parse st = (c, L) and ct = (i, ct_0)
  FSSR.Punc(L[i], ct_0) → L[i]
                          ▷ L[i] is updated
  st ← (c, L)
  return st

SR.Enc(st, pt):
  parse st = (c, L)
  if c = 0 then
      c ← Δ
      FSSR.Init(1^λ) → s
      L ← (L, s)
                          ▷ add a new FSSR state in L
  end if
  c ← c − 1
  set ℓ to the length of L
  FSSR.Enc(L[ℓ], pt) → (L[ℓ], ct_0)
                          ▷ L[ℓ] is updated
  st ← (c, L)
  ct ← (ℓ, ct_0)
  return (st, ct)
```

Figure 9 Post-compromise secure self-ratchet from forward secure self-ratchet.

3.3 Constructions

We provide a generic construction SR from an FSSR scheme[6] providing forward security. For every Δ, we create a new structure with forward security and store it. Given a scheme FSSR offering only forward security, we construct SR as in Fig. 9.

▶ **Theorem 8.** *Let $n(\lambda)$ and $\Delta(\lambda)$ be polynomially bounded positive integer functions of a security parameter λ. Let* opt *be either $\perp$ or* {replay}*. Let* FSSR *be a self-ratcheted scheme which is* IND-(opt ∪ {noPCS}) *secure at level Δ. Then,* SR *(in Fig. 9, with parameter Δ) is a self-ratcheted scheme which is* IND-opt *secure at level n with delay Δ.*

Proof. Let opt be either $\perp$ or {replay} and $\mathcal{B}$ be an IND-opt adversary against SR with delay Δ. Assume that $\mathcal{B}$ queries at most q encryption and challenge queries. Then, we can construct an IND-(opt ∪ {noPCS}) adversary $\mathcal{A}$ against FSSR at level Δ as shown on Fig. 10.

By the construction, SR generates a new state of FSSR for each Δ encryptions. The adversary $\mathcal{A}$ therefore simulates the IND-opt security game with delay Δ while trying to replace Δ ciphertexts by the ciphertexts that the adversary is challenging. If the oracle Challenge′ does not abort the game, the adversary $\mathcal{A}$ can correctly guess b if $\mathcal{B}$ can correctly guess it. The probability that the game is not aborted by Challenge′ is about Δ/q. Then, the advantage of $\mathcal{A}$ is

$$\mathsf{Adv}^{\mathsf{IND}^{\mathsf{FSSR},(\mathsf{opt}\cup\{\mathsf{noPCS}\})}}_{\Delta,\cdot,\lambda}(\mathcal{A}) = \frac{1}{\lceil q/\Delta \rceil}\mathsf{Adv}^{\mathsf{IND}^{\mathsf{SR},\mathsf{opt}}}_{n,\Delta,\lambda}(\mathcal{B})$$

Since q is polynomially bounded and $\Delta \geq 1$, if $\mathsf{Adv}^{\mathsf{IND}^{\mathsf{FSSR},(\mathsf{opt}\cup\{\mathsf{noPCS}\})}}_{\Delta,\cdot,\lambda}(\mathcal{A})$ is negligible, then $\mathsf{Adv}^{\mathsf{IND}^{\mathsf{SR},\mathsf{opt}}}_{n,\Delta,\lambda}(\mathcal{B})$ is negligible too. Hence, SR is IND-opt secure at level n with delay Δ if FSSR is IND-(opt ∪ {noPCS}) secure at level Δ. ◀

[6] FSSR means FS-*secure self-ratcheted scheme.*

$\mathcal{A}^{\mathsf{OEnc},\mathsf{ODec},\mathsf{Challenge},\mathsf{OPunc},\mathsf{OExp}}(1^\lambda)$:

```
idx ←$ {1, ..., ⌈q/Δ⌉}
SR.Init(1^λ) → st
Active, Revealed ← ∅
challenged ← false
AfterExp ← Δ
B^{OEnc', ODec', Challenge', OPunc', OExp'}(1^λ) → b'
return b'
```

Subroutine $\mathsf{OEnc}'(\mathsf{pt})$:

```
if |Active| ≥ n then return ⊥
SR.Enc(st, pt) → (st, ct)
parse ct = (ℓ, ct_0)
if ℓ = idx then
    OEnc(pt) → ct_0
end if
ct ← (ℓ, ct_0)
Active ← Active ∪ {ct}
Revealed ← Revealed ∪ {ct}
AfterExp ← AfterExp + 1
return ct
```

Subroutine $\mathsf{ODec}'(\mathsf{ct})$:

```
if ct ∈ Active − Revealed then
    return ⊥
end if
parse ct = (ℓ, ct_0)
if ℓ = idx then
    ODec(ct_0) → pt
else
    SR.Dec(st, ct) → (st, pt)
end if
if "replay" ∈ opt then OPunc'(ct)
return pt
```

Subroutine $\mathsf{OPunc}'(\mathsf{ct})$:

```
parse ct = (ℓ, ct_0)
if ℓ = idx then
    OPunc(ct_0)
else
    SR.Punc(st, ct) → st
end if
Active ← Active − {ct}
Revealed ← Revealed − {ct}
return
```

Subroutine $\mathsf{Challenge}'(\mathsf{pt})$:

```
if |Active| ≥ n or AfterExp < Δ then
    return ⊥
end if
parse st = (c, L)
if (c ≠ 0 or |L| ≠ idx − 1) and (c = 0 or
    |L| ≠ idx) then
    abort the game
end if
SR.Enc(st, pt) → (st, ct)
Challenge(pt) → ct
Active ← Active ∪ {ct}
AfterExp ← AfterExp + 1
challenged ← true
return ct
```

Subroutine $\mathsf{OExp}'()$:

```
parse st = (c, L)
if |L| ≥ idx then
    OExp() → st'
    if st' = ⊥ then return ⊥
    L[idx] ← st'
end if
AfterExp ← 0
return (c, L)
```

Figure 10 FS adversary for FSSR based on an adversary for SR.

Optimization

Our SR scheme can obviously be optimized for storage. For each state $L[i]$, we can add a counter of active ciphertexts with $L[i]$ which is incremented by Enc and decremented by Punc (after checking that decryption works). Then, when the counter becomes 0, $L[i]$ can be erased.

Another convenient optimization holds when the application wants to operate bulk puncturing of too old ciphertexts. This implies to erase all first $L[i]$. It is quite compatible with recent policies of session resumption: a session which is too old cannot be resumed.

$\underline{\mathsf{FSSR.Init}(1^\lambda)}$:
$\mathsf{PPRF.Setup}(1^\lambda) \to k_{\mathsf{PPRF}}$
$\mathsf{st} \leftarrow (k_{\mathsf{PPRF}}, 0)$
return st

$\underline{\mathsf{FSSR.Punc}(\mathsf{st}, \mathsf{ct})}$:
parse $\mathsf{st} = (k_{\mathsf{PPRF}}, \mathsf{cnt})$
parse $\mathsf{ct} = (\mathsf{cnt}', \mathsf{ct}_0)$
$\mathsf{pt} \leftarrow \mathsf{FSSR.Dec}(\mathsf{st}, \mathsf{ct})$
if $\mathsf{pt} = \perp$ **then return** $\perp$
$\mathsf{PPRF.Punc}(k_{\mathsf{PPRF}}, \mathsf{cnt}') \to k_{\mathsf{PPRF}}$
if $k_{\mathsf{PPRF}} = \perp$ **then return** $\perp$
$\mathsf{st} \leftarrow (k_{\mathsf{PPRF}}, \mathsf{cnt})$
return st

$\underline{\mathsf{FSSR.Enc}(\mathsf{st}, \mathsf{pt})}$:
parse $\mathsf{st} = (k_{\mathsf{PPRF}}, \mathsf{cnt})$
$\kappa \leftarrow \mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, \mathsf{cnt})$
if $\kappa = \perp$ **then return** $\perp$
$\mathsf{ct}_0 \leftarrow \mathsf{AEAD.Enc}(\kappa, \mathsf{cnt}, \mathsf{pt})$
$\mathsf{ct} \leftarrow (\mathsf{cnt}, \mathsf{ct}_0)$
$\mathsf{st} \leftarrow (k_{\mathsf{PPRF}}, \mathsf{cnt} + 1)$
return st, ct

$\underline{\mathsf{FSSR.Dec}(\mathsf{st}, \mathsf{ct})}$:
parse $\mathsf{st} = (k_{\mathsf{PPRF}}, \mathsf{cnt})$
parse $\mathsf{ct} = (\mathsf{cnt}', \mathsf{ct}_0)$
$\kappa \leftarrow \mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, \mathsf{cnt}')$
if $\kappa = \perp$ **then return** $\perp$
$\mathsf{pt} \leftarrow \mathsf{AEAD.Dec}(\kappa, \mathsf{cnt}', \mathsf{ct})$
return pt

Figure 11 FS-secure SR.

3.4 FS-Secure Self-Ratcheted Scheme (from AGJ)

We adapt[7] the generic construction from Aviram et al. [2] based on a puncturable PRF denoted as PPRF. We use authenticated encryption with associated data AEAD = (Gen, Enc, Dec). (In our notation, the second input to Enc and Dec is the associated data i.e. the header to be authenticated.) The construction is in Fig. 11.

AGJ presented two possible PPRF constructions. One is based on the Camenisch-Lysyanskaya RSA accumulator [5]. The other is based on a tree structure.

RSA-based PPRF

The RSA-based construction uses a PPRF key of linear size in terms of the number of encryptions and can only handle a polynomial number of encryptions. This is the total number of encryptions, i.e. not only the ones remaining active. We give the construction in Fig. 12, using a random oracle H and the list of first odd primes $(p_1, \dots, p_m)$. In the original paper [2], the authors have shown that the above construction is a secure PPRF in the random oracle model, under the strong RSA assumption. The PPRF key is of size $2\lambda + m$. However, the N part of the key can be set as a domain parameter which is common to many keys.

In our construction, the device only needs to encrypt Δ messages per PPRF key. Hence, we can set $m = \Delta$ in the above PPRF, meaning that the FSSR has states of size $\lambda + \Delta + \log_2 \Delta$ plus λ bits of common parameter N. Finally, our SR has states of size

$$\ell = \frac{n}{\Delta}(\lambda + \Delta + \log_2 \Delta) + \log_2 \Delta + \lambda \tag{3}$$

We can see that $\frac{\ell\Delta}{n}$ is at least linear in λ, hence super-logarithmic. Compared to (1), we are within a factor close to 2 to the lower bound.

[7] The only change is the separation between Dec and Punc.

$\mathsf{PPRF.Setup}(1^\lambda)$:
- generate an RSA modulus $N = pq$ of length λ using safe primes
- erase p and q
- pick $g \in \mathbf{Z}_N$ at random
- $r \leftarrow (0, 0, \dots, 0) \in \{0,1\}^m$
- $k_{\mathsf{PPRF}} \leftarrow (N, g, r)$
- **return** k_{PPRF}

$\mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, x)$:
- parse $k_{\mathsf{PPRF}} = (N, g, r)$
- **if** $r_x = 1$ **then return** $\perp$
- $P_x \leftarrow \prod_{i=1}^{m} p_i^{r_i \cdot 1_{i \neq x}}$
- $y \leftarrow H(g^{P_x} \bmod N)$
- **return** y

$\mathsf{PPRF.Punc}(k_{\mathsf{PPRF}}, x)$:
- parse $k_{\mathsf{PPRF}} = (N, g, r)$
- **if** $r_x = 1$ **then return** $\perp$
- $g \leftarrow g^{p_x} \bmod N$
- $r_x \leftarrow 1$
- $k_{\mathsf{PPRF}} \leftarrow (N, g, r)$
- **return** k_{PPRF}

Figure 12 RSA-based PPRF.

Tree-based PPRF

The tree-based constructions is formed with two functions G_0 and G_1 from $\{0,1\}^\lambda$ to itself, which we extend to functions G_z for every binary word z by $G_{xy}(L) = G_y(G_x(L))$. Then, the PPRF defines a binary tree of depth d which is partially labeled. The PPRF key is a set of (x, L) pairs where x is a binary word (hence a node in the binary tree) and L is its label in $\{0,1\}^\lambda$. Initially, the key consists of the label of the root ε. To evaluate on x, one should find a labeled node (y, L) such that y is a prefix of x, write $x = yz$, and return $G_z(L)$. The interface of the PPRF only takes d-bit input x (i.e. leaves), but our evaluation is defined for every node. To puncture a leaf x, one should find this y again and replace (y, L) from the key by the list of (x', L') with $x' = yz_1 \cdots z_{i-1}\bar{z}_i$ and L' being the evaluation on x', where $z_1 \cdots z_{|z|}$ is the binary expansion of z and $\bar{z}_i$ is the bit complement of z_i. Hence, a PPRF key is an anti-chain with no siblings. In the worst case, it could inflate by d pairs at every puncture, but the maximum length is of 2^{d-1} pairs.

Same as the RSA construction, one only needs to evaluate $2^d = \Delta$ leaves. In the worst case, a PPRF key has length $2^{d-1} \times d\lambda$ which is $\frac{1}{2}\lambda\Delta \log_2 \Delta$. Hence, the FS-secure self-ratcheted scheme has states of size bounded by $\frac{1}{2}\lambda\Delta\log_2\Delta + \log_2\Delta$. Finally, our secure self-ratcheted scheme has states of size

$$\ell = \frac{n}{\Delta}\left(\frac{1}{2}\lambda\Delta\log_2\Delta + \log_2\Delta\right) + \log_2\Delta$$

which is larger than with the RSA-based method.

3.5 Experimental Results

We instantiate an SR based on FSSR with the RSA-based PPRF. We assumed that the same RSA modulus is used for all PPRF keys, the RSA modulus so is precomputed and given as a parameter to SR. Hence, the cost of setting up the RSA modulus is not covered in our analysis. For H and AEAD, we used SHA-256 and AES-GCM.

Our experiment was done on a machine with the AMD Opteron 8354 processor and 128 GB of RAM by using the SageMath version 8.7. We picked a common 2048-bit RSA modulus.

We tried many values for Δ from $\Delta = 100$ to $\Delta = 10\,000$ by steps of 100. We measured the worst case complexity of an SR.Enc encryption, which is actually the very first one when nothing is punctured and which includes FSSR.Init, as well as the best case complexity of SR.Enc, which is the very last one after all other values have been punctured. For accuracy, we did it 1000 times for each Δ and took the average. The results are plotted in Fig. 13.

On the plot, we added the total state size divided by the total number n of encryptions as it goes to infinity. This is essentially $\frac{\ell}{n}$ with ℓ given by Eq.(3). As we can see, the execution time grows linearly with Δ while $\frac{\ell}{n} - 1$ is inverse proportional to Δ.

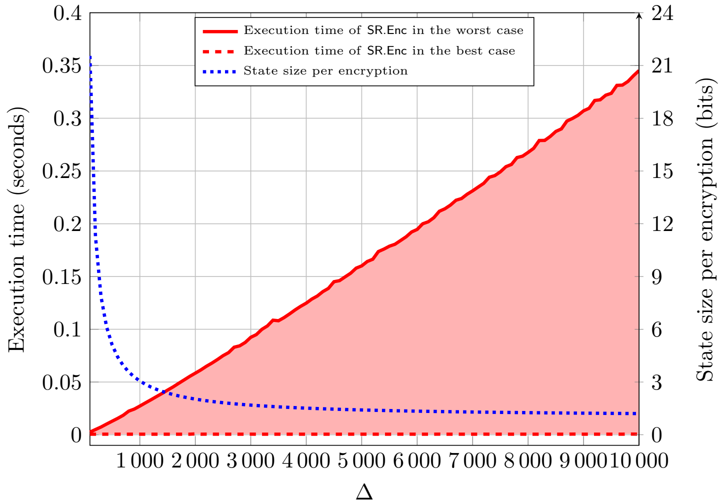


Figure 13 The execution time of SR.Enc in the worst/best case and the state size divided by the number of encryptions with 2048-bit RSA modulus.

4 Bipartite Ratcheted Communication

4.1 Definitions

We consider a ratcheted scheme $S = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ following the syntax

- $S.\mathsf{Gen}(1^\lambda) \to (\mathsf{st}_A, \mathsf{st}_B)$ (generate a pair of states)
- $S.\mathsf{Enc}(\mathsf{st}) \to (\mathsf{st}', \mathsf{pt}, \mathsf{ct})$ (update the state while producing a pt/ct pair)
- $S.\mathsf{Dec}(\mathsf{st}, \mathsf{ct}) \to (\mathsf{st}', \mathsf{pt})$ (update the state while decrypting ct)

To avoid defining a general correctness and security for ratcheted schemes, which is quite lengthy and complicated, we only adopt a definition matching a particular case of our interest. This is the case when one participant Alice desperately tries to reach her counterpart Bob by consistently sending messages without receiving any response, while Bob actually acknowledges for the receipt of every message from Alice but his acknowledgments somehow never make it through. (See Fig. 15.)

▶ **Definition 9.** *A* ***simple ratcheted scheme*** *is a primitive S defined by $S = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ which is* ***n-correct*** *in the sense that the game in Fig. 14 never returns 1.*

In this communication pattern, protocols such as PR [14], JS [12], JMM [13], and DV [8] have growing states. We can clearly see it on the implementation results by Caforio et al. [4]. Protocols such as Signal [15] or ACD [1] keep constant-size states but offer no post-compromise security in our communication pattern. In fact, in ACD, Alice keeps sending messages in the same "epoch" (following the terminology of ACD [1]) using the forward secure scheme called FS in ACD, while Bob receives those messages from an old epoch (for him) and keeps sending messages in his own epoch, using FS as well. Since the FS scheme is deterministic, it

$S.\mathsf{Gen}(1^\lambda) \to (\mathsf{st}_0^A, \mathsf{st}_0^B)$
for $i = 1$ to n **do**
  $S.\mathsf{Enc}(\mathsf{st}_{i-1}^A) \to (\mathsf{st}_i^A, \mathsf{pt}'_i, \mathsf{ct}'_i)$
  $S.\mathsf{Dec}(\mathsf{st}_{i-1}^B, \mathsf{ct}'_i) \to (x, \mathsf{pt}'_i)$
  $S.\mathsf{Enc}(x) \to (\mathsf{st}_i^B, \mathsf{pt}_i, \mathsf{ct}_i)$
end for
$x \leftarrow \mathsf{st}_n^A$
for $i = 1$ to n **do**
  $S.\mathsf{Dec}(x, \mathsf{ct}_i) \to (x, \mathsf{pt})$
  if $\mathsf{pt} \neq \mathsf{pt}_i$ **then return** 1
end for
return 0

Figure 14 Correctness game for a simple ratcheted scheme of level-n.

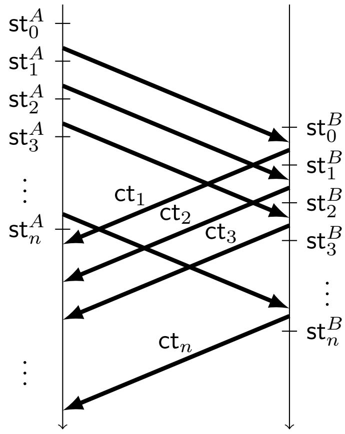


Figure 15 Simulation of the level-n correctness game.

offers no post-compromise security. In ACD-PK, there is an extra public-key encryption but the decryption key remains constant within the same epoch. Hence, exposing st_1^A is enough to decrypt all ciphertexts in both ACD and ACD-PK.

Post-compromise security should make impossible to decrypt ct_m which was released after having ratcheted Δ times both participants after the last state exposure which revealed $\mathsf{st}_{m-\Delta}^A$ and $\mathsf{st}_{m-\Delta}^B$. For instance, with $\Delta = 1$ and $m = 2$, it should be impossible on Fig. 15 to compute pt_2 from $(\mathsf{st}_1^A, \mathsf{st}_1^B, \mathsf{ct}_1, \mathsf{ct}_2)$. This is formalized by the following definition.

▶ **Definition 10.** *Let $n(\lambda)$ and $\Delta(\lambda)$ be polynomially bounded positive integer functions of a security parameter λ. For a simple ratcheted scheme S which is n-correct, we define the game in Fig. 16 with parameters $m \leq n$ and $\Delta > 0$: We say that **S with level n is Δ-secure** if for any PPT adversary $\mathcal{A}$, $\lambda \mapsto \max_{1 \leq m \leq n} \Pr[\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A}) \to 1]$ is a negligible function.*

4.2 Impossibility Result

▶ **Theorem 11.** *For every integer n, ℓ, $\Delta > 0$ and any n-correct simple ratcheted scheme S following Def. 9, and such that $(\mathsf{st}_A, \mathsf{st}_B)$ belongs to a space of size bounded by 2^ℓ, there exists an adversary of low complexity having advantage*

$$\Pr[\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A}) \to 1] > \frac{1}{4n^2} 2^{-2^{\frac{\ell+1}{\lfloor n/\Delta \rfloor}}}$$

in the security game of Def. 10.

Proof. We construct a SEQ protocol P as shown in Fig. 17. If S is n-correct (in the sense of Def. 9), then this new scheme P is correct to level n (in the sense of Def. 1). This comes

$\mathsf{OW}_{m,\Delta,\lambda}$:

$S.\mathsf{Gen}(1^\lambda) \to (\mathsf{st}_0^A, \mathsf{st}_0^B)$
for $i = 1$ to m **do**
  $S.\mathsf{Enc}(\mathsf{st}_{i-1}^A) \to (\mathsf{st}_i^A, \mathsf{pt}_i', \mathsf{ct}_i')$
  $S.\mathsf{Dec}(\mathsf{st}_{i-1}^B, \mathsf{ct}_i') \to (x, \mathsf{pt}_i')$
  $S.\mathsf{Enc}(x) \to (\mathsf{st}_i^B, \mathsf{pt}_i, \mathsf{ct}_i)$
end for
$\mathcal{A}(1^\lambda, \mathsf{st}_{m-\Delta}^A, \mathsf{st}_{m-\Delta}^B, \mathsf{ct}_1, \ldots \mathsf{ct}_m) \to x$
return $1_{x=\mathsf{pt}_m}$

Figure 16 OW game for a simple ratcheted scheme.

$P.\mathsf{Gen}(1^\lambda) \to \mathsf{st}$:

$S.\mathsf{Gen}(1^\lambda) \to (\mathsf{st}_A, \mathsf{st}_B)$
$\mathsf{st} \leftarrow (\mathsf{st}_A, \mathsf{st}_B)$
return st

$P.\mathsf{Dec}(\mathsf{st}, \mathsf{ct}) \to (\mathsf{st}', \mathsf{pt})$:

parse $\mathsf{st} = (\mathsf{st}_A, \mathsf{st}_B)$
$S.\mathsf{Dec}(\mathsf{st}_A, \mathsf{ct}) \to (\mathsf{st}_A', \mathsf{pt})$
$\mathsf{st}' \leftarrow (\mathsf{st}_A', \mathsf{st}_B)$
return $(\mathsf{st}', \mathsf{pt})$

$P.\mathsf{Enc}(\mathsf{st}) \to (\mathsf{st}', \mathsf{pt}, \mathsf{ct})$:

parse $\mathsf{st} = (\mathsf{st}_A, \mathsf{st}_B)$
$S.\mathsf{Enc}(\mathsf{st}_A) \to (\mathsf{st}_A', \mathsf{pt}', \mathsf{ct}')$
$S.\mathsf{Dec}(\mathsf{st}_B, \mathsf{ct}') \to (\mathsf{st}_B', \mathsf{pt}'')$
$S.\mathsf{Enc}(\mathsf{st}_B') \to (\mathsf{st}_B'', \mathsf{pt}, \mathsf{ct})$
$\mathsf{st}' \leftarrow (\mathsf{st}_A', \mathsf{st}_B'')$
return $(\mathsf{st}', \mathsf{pt}, \mathsf{ct})$

Figure 17 Simple ratchet S to SEQ.

from a direct translation of definitions. Furthermore, any uniform adversary against P (in the sense of Def. 2) translates into an adversary against S in the sense of Def. 10: guess m then given $(\mathsf{st}_{m-\Delta}^A, \mathsf{st}_{m-\Delta}^B)$ the adversary decrypts ct_m. We conclude by applying Cor. 4. ◀

5 Conclusion

We defined a self-encryption mechanism involving a device which encrypts a secret message for herself to use in the future. We are interested in security when the state of a device in such settings leaks causing the leakage of the secret message. We started giving some instances where self-ratcheting finds applications in cloud storage, when a client encrypts files to be stored, and in 0-RTT session resumption, when a server encrypts a resumption key to be kept by the client. Unlike previous works which focused on forward security and resistance to replay attacks, we studied how to add post-compromise security, as well.

We first proved that post-compromise security implies a super-linear state size in terms of the number of ciphertexts which can still be decrypted by the state. We then give formal definitions of self-ratchet. We finally showed how to design a secure scheme satisfying our bound on the state size.

Furthermore, we showed that our results on the growth of state size matches with existing secure bidirectional secure messaging applications. Given the fact that the messaging applications provide different level of PCS, we observed that there exist some protocols such as ACD without growing state size. It is due to the fact that the protocol is secure with a weaker notion of PCS which could allow constant-size states. It would be interesting to investigate weaker PCS notions in self-encryption applications such as cloud storage or 0-RTT.

References

1 Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol. In *Advances in Cryptology – EUROCRYPT 2019*, LNCS. Springer, 2019. doi:10.1007/978-3-030-17653-2_5.

2 Nimrod Aviram, Kai Gellert, and Tibor Jager. Session Resumption Protocols and Efficient Forward Security for TLS 1.3 0-RTT. In *Advances in Cryptology – EUROCRYPT 2019*, LNCS. Springer, 2019. doi:10.1007/978-3-030-17656-3_5.

3 Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key Homomorphic PRFs and Their Applications . In *Advances in Cryptology – CRYPTO 2013*, LNCS. Springer, 2013. doi:10.1007/978-3-642-40041-4_23.

4 Andrea Caforio, F. Betül Durak, and Serge Vaudenay. Beyond security and efficiency: On-demand ratcheting with security awareness. In *Public Key Cryptography – PKC 2021*, LNCS. Springer, 2021. Full version: Cryptology ePrint Archive, Report 2019/965 https://eprint.iacr.org/2019/965. doi:10.1007/978-3-030-75248-4_23.

5 Jan Camenisch and Anna Lysyanskaya. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *Advances in Cryptology - CRYPTO 2002*, LNCS. Springer, 2002. doi:10.1007/3-540-45708-9_5.

6 Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. On post-compromise security. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 164–178, June 2016. Full version: Cryptology ePrint Archive, Report 2016/221 https://eprint.iacr.org/2016/221. doi:10.1109/CSF.2016.19.

7 David Derler, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Bloom Filter Encryption and Applications to Efficient Forward-Secret 0-RTT Key Exchange. In *Advances in Cryptology – EUROCRYPT 2018*, LNCS. Springer, 2018. doi:10.1007/s00145-021-09374-3.

8 F. Betül Durak and Serge Vaudenay. Bidirectional Asynchronous Ratcheted Key Agreement with Linear Complexity. In *Advances in Information and Computer Security – IWSEC 2019*, LNCS. Springer, 2019. Full version: Cryptology ePrint Archive, Report 2018/889 https://eprint.iacr.org/2018/889. doi:10.1007/978-3-030-26834-3_20.

9 Adam Everspaugh, Kenneth Paterson, Thomas Ristenpart, and Sam Scott. Key Rotation for Authenticated Encryption. In *Advances in Cryptology – CRYPTO 2017*, LNCS. Springer, 2017. doi:10.1007/978-3-319-63697-9_4.

10 Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-RTT Key Exchange with Full Forward Secrecy. In *Advances in Cryptology – EUROCRYPT 2017*, LNCS. Springer, 2017. doi:10.1007/978-3-319-56617-7_18.

11 Ralph Holz, Johanna Amann, Abbas Razaghpanah, and Narseo Vallina-Rodriguez. The Era of TLS 1.3: Measuring Deployment and Use with Active and Passive Methods. *CoRR*, 2019. URL: http://arxiv.org/abs/1907.12762.

12 Joseph Jaeger and Igors Stepanovs. Optimal Channel Security Against Fine-Grained State Compromise: The Safety of Messaging . In *Advances in Cryptology – CRYPTO 2018*, LNCS. Springer, 2018. doi:10.1007/978-3-319-96884-1_2.

13 Daniel Jost, Ueli Maurer, and Marta Mularczyk. Efficient Ratcheting: Almost-Optimal Guarantees for Secure Messaging. In *Advances in Cryptology – EUROCRYPT 2019*, LNCS. Springer, 2019. doi:10.1007/978-3-030-17653-2_6.

14 Bertram Poettering and Paul Rösler. Towards bidirectional ratcheted key exchange. In *Advances in Cryptology – CRYPTO 2018*, LNCS. Springer, 2018. doi:10.1007/978-3-319-96884-1_1.

15 Open Whisper Systems. Signal protocol library for Java/Android. GitHub repository https://github.com/WhisperSystems/libsignal-protocol-java, 2017.