A Note on the Unforgeability Hierarchy of Non-Interactive Threshold Signatures, with a Mechanisation in EasyCrypt

João Diogo Duarte

University of Porto (FCUP) and INESC TEC, Portugal

Abstract. Bellare et al. [BCK+22] define a hierarchy of unforgeability notions for non-interactive threshold signatures, TS-UF-0 to TS-UF-4 and TS-SUF-2 to TS-SUF-4. They prove that each notion implies the weaker ones and that TS-UF-0 implies TS-UF-1 with a loss of $\binom{ns}{t-1}$ for ns servers and threshold t, and they separate every other pair of notions. Cremers, Peltonen and Zhao [CPZ24] showed that TS-UF-4 does not imply TS-UF-3, and Bellare, Tessaro and Zhu revised the definitions in response [BTZ22]. We give machine-checked proofs in EasyCrypt of the revised hierarchy, of the loose implication from TS-UF-0 to TS-UF-1, and of the attacks behind the separations. We also prove that the loss of $\binom{ns}{t-1}$ is optimal for an implication that holds for every scheme, up to the security of a signature scheme.

Keywords: Threshold signatures · unforgeability · security notions · machine-checked proofs · EasyCrypt

1 Introduction

A threshold signature scheme shares a signing key among ns servers, so that any t of them can sign and fewer cannot. In a non-interactive scheme, a server answers a signing request with a single reply, possibly after a round of preprocessing that does not depend on the message to be signed, as in FROST [KG20]. Bellare et al. [BCK+22] give a syntax for these schemes and a hierarchy of unforgeability notions, TS-UF-0 to TS-UF-4, with strong variants TS-SUF-2 to TS-SUF-4. All the notions are defined by one game and differ only in which forgeries count as trivial. Bellare et al. prove on paper that each notion implies the weaker ones and that TS-UF-0 implies TS-UF-1 with a loss of $\binom{ns}{t-1}$. They show that no other implications hold.

Cremers, Peltonen and Zhao (CPZ) [CPZ24] found that under these definitions TS-UF-4 does not imply TS-UF-3, and TS-SUF-4 does not imply TS-SUF-3. Bellare, Tessaro and Zhu (BTZ) revised the definitions on ePrint [BTZ22][1]. In the preprocessing round a server issues a token, and a signing request echoes a token back to each server that it names. The revision requires a server to refuse a request that echoes to it a token that it did not issue, and adds the same check to the signing oracle of the game. CPZ note that the revision rules out their counterexample [CPZ24, Section 5]. The security of FROST [KG20] is stated in these notions [BCK+22]. We therefore state the game of the revision in EasyCrypt [BGHZ11], and prove the implications and the attacks behind the separations for this game.

Theorem 1 gives the revised hierarchy for every scheme and every adversary. Its proof uses the check in the signing oracle and not the check by the servers, and Theorem 2 shows that the check by the servers can replace the check in the oracle. Theorem 3 gives the loose implication from TS-UF-0 to TS-UF-1, and Theorem 4 shows that its loss of $\binom{ns}{t-1}$ is optimal for an implication that holds for every scheme, up to the security of a signature scheme. Theorem 5 gives the attacks behind the separations. For the separation of TS-SUF-2 from TS-UF-3, the formalisation needed a change to the verification of BTZ's scheme and a different attack.

The development is available at https://codeberg.org/JoaoDiogoDuarte/threshold-hierarchy. Appendix A gives the main definitions in EasyCrypt, and Appendix B gives the lemmas behind the numbered results.

E-mail: joao@diogoduarte.pt (João Diogo Duarte)

[1] The CRYPTO version [BCK+22] has Crites, Komlo and Maller as further authors. We cite the ePrint revision of 4 December 2024, the latest version in the archive at the time of writing.

This work is licensed under a "CC BY 4.0" license.

Date of this document: 2026-11-19.

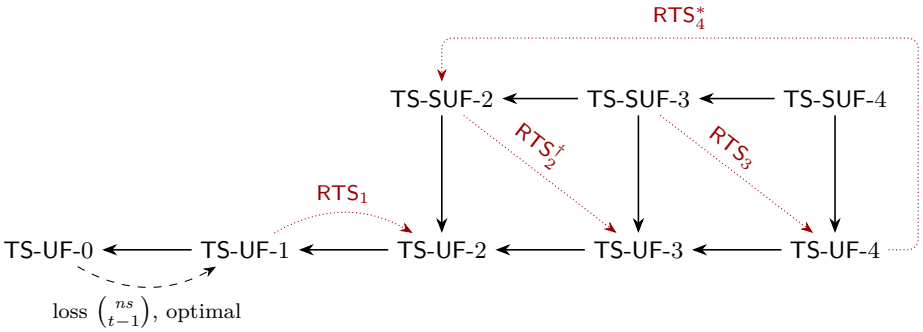

Figure 1: The relations between the notions. A solid arrow from X to Y is an implication of Theorem 1, and the dashed arrow is the loose implication of Theorem 3, whose loss is optimal up to the security of DS by Theorem 4. A dotted arrow from X to Y is a separation by the labelled scheme, which is broken at Y by Theorem 5 and secure for X by Proposition 2, which we do not mechanise. The variant RTS^*_4 is defined in Section 6. The mark † indicates the separation for which Remarks 1 and 2 change BTZ's scheme and attack, and which needs Proposition 2 for the changed scheme. Every other relation follows from these by transitivity.

2 The Games

We recall the syntax and the game of BTZ's revision [BTZ22]. A scheme TS has ns servers, numbered from 1 to ns, a threshold $t \in [1..ns]$, and a leader who coordinates signing. Key generation Kg returns a verification key vk, public auxiliary information aux, and one secret key for each server. In the preprocessing round, server i runs SPP and sends the resulting token pp to the leader. To sign a message, the leader builds a leader request lr. The request names the message $lr.\mathsf{msg}$, a set $lr.\mathsf{SS}$ of servers and, for each $i \in lr.\mathsf{SS}$, a token $lr.\mathsf{PP}(i)$ that the leader echoes back to server i. Server i answers by running PS, which returns a partial signature or $\perp$. The leader aggregates the partial signatures into a signature, and Vf verifies it. In Figure 2, TS.HF is the family of hash functions of the scheme, and $[h]$ marks an algorithm with oracle access to the function h that line 2 draws from it.

In the game of Figure 2, the adversary corrupts a set CS of fewer than t servers, plays the leader, and queries the other servers, which form the set HS, through PPO and PSignO. The game records the tokens PP_i that server i issued and the requests L that passed the checks of PSignO. These, and the sets S_1 and S_2, are empty when Init returns, and a failed **Require:** makes the adversary lose at once. The set $S_1(M)$ holds the honest servers that returned a partial signature for a request with message M, and $S_2(lr)$ those that returned one for the request lr. When Fin runs, it derives $S_3(lr)$ and $S_4(lr)$ for each request in L. The adversary wins if its signature verifies and its forgery is not trivial.

We call i the level of TS-UF-i and TS-SUF-i. The table in Figure 2 gives the trivial-forgery predicates. At level 0, a forgery on M is trivial as soon as one honest server signed M. At level 1, it is trivial when at least $t - |CS|$ honest servers signed M, possibly under different requests. Level 2 asks for $t - |CS|$ of them under a single request for M. Level 3 asks in addition that the request was answered by every honest server that it names and that issued the token echoed to it. This condition is $S_3(lr) \subseteq S_2(lr)$, and the converse inclusion holds in the revised game by Lemma 1. At level 4, the request must have been answered by every honest server that it names. The strong notions are defined for schemes with a strong verification algorithm SVf, under which at most one signature is valid for each pair (vk, lr). A forgery is then trivial for TS-SUF-i only if, in addition, SVf accepts its signature for the request, so a new signature for an answered request counts as a forgery. We write $\mathbf{Adv}^{\text{ts-uf-}i}_{\mathsf{TS}}(\mathcal{A}) = \Pr[\mathbf{G}^{\text{ts-uf-}i}_{\mathsf{TS}}(\mathcal{A}) = 1]$ and $\mathbf{Adv}^{\text{ts-suf-}i}_{\mathsf{TS}}(\mathcal{A}) = \Pr[\mathbf{G}^{\text{ts-suf-}i}_{\mathsf{TS}}(\mathcal{A}) = 1]$ for the probability that the adversary $\mathcal{A}$ wins. A scheme is TS-UF-i-secure when $\mathbf{Adv}^{\text{ts-uf-}i}_{\mathsf{TS}}(\mathcal{A})$ is small for every adversary $\mathcal{A}$ that uses reasonable resources, and likewise for TS-SUF-i.

Line 11 refuses a request lr to server i unless $lr.\mathsf{PP}(i)$ is a token that server i issued. In BTZ's syntax a request need not contain tokens, and BTZ call a scheme whose requests contain them an echo scheme. The revision requires each server of an echo scheme to refuse a request that echoes to it a token that it did not issue, and we call this the server check. Server i then keeps a set $st_i.\mathsf{PP}$ of the tokens that it issued, and PS returns $\perp$ when $lr.\mathsf{PP}(i) \notin st_i.\mathsf{PP}$. BTZ apply line 11 only to echo schemes, and define level 3 for

INIT(CS):
1 **Require:** $CS \subseteq [1..ns]$ and $|CS| < t$
2 $h \leftarrow\$ \mathsf{TS.HF}$; $(vk, aux, sk_1, \ldots, sk_{ns}) \leftarrow\$ \mathsf{Kg}[h]$
3 $HS \leftarrow [1..ns] \setminus CS$
4 For $i \in HS$ do
5 $\quad st_i.sk \leftarrow sk_i$; $st_i.vk \leftarrow vk$; $st_i.aux \leftarrow aux$
6 Return $vk, aux, \{sk_i\}_{i \in CS}$

PPO(i):
7 **Require:** $i \in HS$
8 $(pp, st_i) \leftarrow\$ \mathsf{SPP}[h](st_i)$; $\mathrm{PP}_i \leftarrow \mathrm{PP}_i \cup \{pp\}$; Return pp

PSIGNO(i, lr):
9 $M \leftarrow lr.\mathsf{msg}$
10 **Require:** $lr.\mathsf{SS} \subseteq [1..ns]$ and $M \in \{0,1\}^*$ and $i \in HS \cap lr.\mathsf{SS}$
11 **Require:** $lr.\mathsf{PP}(i) \in \mathrm{PP}_i$ // echo schemes
12 $L \leftarrow L \cup \{lr\}$; $(psig, st_i) \leftarrow\$ \mathsf{PS}[h](lr, i, st_i)$
13 If $psig \neq \bot$ then
14 $\quad S_1(M) \leftarrow S_1(M) \cup \{i\}$; $S_2(lr) \leftarrow S_2(lr) \cup \{i\}$
15 Return $psig$

RO(x):
16 Return $h(x)$

FIN(M, sig):
17 For all $lr \in L$ do
18 $\quad S_3(lr) \leftarrow \{\, i \in HS \cap lr.\mathsf{SS} : lr.\mathsf{PP}(i) \in \mathrm{PP}_i \,\}$; $S_4(lr) \leftarrow HS \cap lr.\mathsf{SS}$
19 If not $\mathsf{Vf}[h](vk, M, sig)$ then return false
20 Return not $\mathsf{tf}_i(M)$ // $\mathbf{G}^{\text{ts-uf-}i}_{\mathsf{TS}}$, $i \leq 1$
21 Return not $\exists\, lr$ $(lr.\mathsf{msg} = M$ and $\mathsf{tf}_i(lr))$ // $\mathbf{G}^{\text{ts-uf-}i}_{\mathsf{TS}}$, $2 \leq i$
22 Return not $\exists\, lr$ $(lr.\mathsf{msg} = M$ and $\mathsf{tsf}_i(lr, vk, sig))$ // $\mathbf{G}^{\text{ts-suf-}i}_{\mathsf{TS}}$, $2 \leq i$

Predicate	Trivial when
$\mathsf{tf}_0(M)$	$S_1(M) \neq \emptyset$
$\mathsf{tf}_1(M)$	$t - \lvert CS\rvert \leq \lvert S_1(M)\rvert$
$\mathsf{tf}_2(lr)$	$t - \lvert CS\rvert \leq \lvert S_2(lr)\rvert$
$\mathsf{tf}_3(lr)$	$\mathsf{tf}_2(lr)$ and $S_2(lr) = S_3(lr)$
$\mathsf{tf}_4(lr)$	$\mathsf{tf}_2(lr)$ and $S_2(lr) = S_4(lr)$
$\mathsf{tsf}_i(lr, vk, sig)$	$\mathsf{tf}_i(lr)$ and $\mathsf{SVf}[h](vk, lr, sig)$

Figure 2: The games $\mathbf{G}^{\text{ts-uf-}i}_{\mathsf{TS}}$ and $\mathbf{G}^{\text{ts-suf-}i}_{\mathsf{TS}}$ of BTZ's revision [BTZ22, Figure 2], with its line numbers, and the trivial-forgery predicates [BTZ22, Figure 3], where i ranges over $\{2, 3, 4\}$ in the last row of the table. The shaded parts are absent from the CRYPTO version [BCK+22].

these schemes alone.

3 The Model in EasyCrypt

We state the games and prove the results in EasyCrypt [BGHZ11], a proof assistant for game-based proofs. In EasyCrypt a scheme and an adversary are modules, collections of probabilistic procedures that may keep state in variables of their own, and a game is a module that calls them. A property of every run of a game is proved in Hoare logic, and a relation between two games in probabilistic relational Hoare logic.

3.1 The scheme and the adversary

The game calls Kg, SPP, PS and Vf, and no other procedure of the scheme. It does not call the algorithms of the leader, since the adversary plays the leader, and it evaluates SVf as an operator that is fixed when the model is instantiated, not by calling the procedure of the scheme. The procedures of the scheme receive no oracle, so they call neither the adversary nor the oracles of the game. They cannot read or write the variables of the games or of the reductions, so a server computes only with its arguments, which include its state, and with the variables of the scheme's own module.

The adversary has two stages. The first returns the set CS and cannot query the oracles. The second receives what INIT returns, queries PPO and PSIGNO, and outputs the forgery. The adversary does not receive the scheme, so it cannot call the procedures of the scheme. It cannot read or write the variables of the scheme, of the games or of a reduction that runs it, so it interacts with the honest servers only through the oracles. We prove Theorems 1 to 3 for an arbitrary scheme and an arbitrary adversary, and assume no axiom on either, so both may sample and keep state, and neither needs to terminate.

3.2 The game

All eight games are one program, which takes the notion as a parameter. The preprocessing oracle returns $\perp$ unless $i \in HS$, and otherwise runs SPP on the stored state of server i, stores the new state and adds the token to PP_i, as in lines 7 and 8. The signing oracle checks lines 10 and 11 in one condition, and the requirement $M \in \{0,1\}^*$ of line 10 holds for every message because the model types messages. If the condition fails, the oracle returns $\perp$. Otherwise it runs PS on the stored state of server i and stores the new state. It then appends lr to L and, if the partial signature is not $\perp$, adds i to $S_1(lr.\mathsf{msg})$ and to $S_2(lr)$, as in lines 12 to 14. When the adversary has returned its forgery, the game computes S_3 and S_4 for the requests in L, runs Vf, and evaluates line 1 and lines 19 to 22. Checking line 1 at the end instead of in Init does not change which runs the adversary wins. The quantifier of lines 21 and 22 ranges over the requests in L. A request outside L has an empty S_2, so it satisfies no predicate of levels 2 to 4 when CS satisfies line 1.

3.3 Differences from BTZ's game

The model differs from Figure 2 in three ways. Firstly, line 11 applies to every scheme, while BTZ apply it only to echo schemes. A request in the model always contains a map of tokens, so before the adversary asks server i to sign, it must obtain a token from server i, even when the scheme ignores tokens. Secondly, there is no random oracle, and a scheme that hashes uses a fixed public function h. Each statement is proved for every fixed h, so each reduction must work for every h and cannot choose or program it. Averaging over h drawn as in line 2 turns each statement into a statement in the random-oracle model in which no reduction programs the oracle. Nguyen, Theodorakis and Warinschi [NTW22] show, in the asymptotic setting, that a fully black-box reduction in the standard model gives a reduction that does not program a common setup. Thirdly, when a condition of PPO or PSignO fails, the oracle returns $\perp$ and the game continues, while in BTZ's game the adversary loses at once. The condition of line 1 is checked when Fin runs, so an adversary that violates it loses in both games. Every other condition depends only on CS, on the tokens that PPO returned and on the query, so an adversary can evaluate it before it queries. An adversary that answers its own failing queries with $\perp$ has the same advantage in both games. Conversely, a win in BTZ's game is also a win in ours, so the two games define the same notions. We argue this on paper and not in EasyCrypt.

3.4 The state of a server

In BTZ's syntax, the state st_i is all that server i keeps between two calls of its algorithms. In particular, two servers share no state, as when they run on separate machines. BTZ make the state an explicit input and output of the algorithms, "maintained by the overlying game" [BTZ22, Section 3.1], and so do we. The game starts server i from a state computed from i, vk, aux and sk_i, as in lines 4 and 5, and passes the state returned by one call to the next call. In the model, PS receives the state and the request but not i, which BTZ's PS receives in line 12, so a server that needs its index, for instance to read $lr.\mathsf{PP}(i)$, reads it from its state.

An EasyCrypt module can also have variables of its own, and the type of a scheme cannot rule them out. A scheme in the model can therefore keep state outside the states st_i, which BTZ's syntax does not allow. For instance, Kg can store the secret keys of all servers in a variable of the scheme, and PS can then read the secret key of every server, whichever server it runs for. We call a scheme pure if its procedures change no variable of its module. A scheme whose module has no variables is pure, and the scheme of the example is not, since its Kg writes a variable. Purity matters only for the meaning of Theorem 3, whose adversary runs the procedures of the scheme.

4 The Hierarchy

Theorem 1 (Hierarchy). *For every scheme* TS *and every adversary* $\mathcal{A}$*,*

$$\mathbf{Adv}^{\text{ts-uf-0}}_{\mathsf{TS}}(\mathcal{A}) \le \mathbf{Adv}^{\text{ts-uf-1}}_{\mathsf{TS}}(\mathcal{A}) \le \mathbf{Adv}^{\text{ts-uf-2}}_{\mathsf{TS}}(\mathcal{A}) \le \mathbf{Adv}^{\text{ts-uf-3}}_{\mathsf{TS}}(\mathcal{A}) \le \mathbf{Adv}^{\text{ts-uf-4}}_{\mathsf{TS}}(\mathcal{A}),$$
$$\mathbf{Adv}^{\text{ts-suf-2}}_{\mathsf{TS}}(\mathcal{A}) \le \mathbf{Adv}^{\text{ts-suf-3}}_{\mathsf{TS}}(\mathcal{A}) \le \mathbf{Adv}^{\text{ts-suf-4}}_{\mathsf{TS}}(\mathcal{A}),$$
$$\mathbf{Adv}^{\text{ts-uf-}i}_{\mathsf{TS}}(\mathcal{A}) \le \mathbf{Adv}^{\text{ts-suf-}i}_{\mathsf{TS}}(\mathcal{A}) \quad \textit{for } i \in \{2,3,4\}.$$

A scheme that is secure for TS-UF-i is therefore secure for TS-UF-j for every $j \leq i$, and likewise for the strong notions. A scheme that is secure for TS-SUF-i is also secure for TS-UF-i. These are the solid arrows of Figure 1. In EasyCrypt the theorem is nine lemmas, one for each inequality, for the arbitrary scheme and adversary of Section 3.1.

4.1 The invariants

All eight games run the same code up to line 19 and differ only in the winning predicate that they check afterwards. Comparing two games therefore reduces to comparing their winning predicates on the state at the end of a run. This comparison needs two facts about that state. Neither follows from the definitions of the sets, because S_1 and S_2 grow as the adversary queries and S_3 and S_4 are computed only when Fin runs. We prove them as invariants, which hold at the start of the game and are preserved by every oracle call [BDL+26].

Lemma 1 (Invariants)**.** *At every point of a run of the game, and for every request lr,*

1. $S_2(lr) \subseteq S_1(lr.\mathsf{msg})$*, and*
2. *every $i \in S_2(lr)$ satisfies $i \in HS \cap lr.\mathsf{SS}$ and $lr.\mathsf{PP}(i) \in \mathrm{PP}_i$.*

Consequently, when Fin *runs, $S_2(lr) \subseteq S_3(lr) \subseteq S_4(lr)$ for every $lr \in L$.*

Proof. Both statements hold when the game starts, since all the sets are empty. A query to PPO only enlarges PP_i, which preserves both statements. A query to PSignO runs line 14 only if lines 10 and 11 have passed, and line 14 adds i to $S_1(lr.\mathsf{msg})$ and to $S_2(lr)$ together. The restriction on the scheme in Section 3.1 ensures that the call to PS in line 12 does not change the sets. For the consequence, the second statement is the condition under which line 18 puts i in $S_3(lr)$, and $S_3(lr) \subseteq S_4(lr)$ holds by definition. □

In EasyCrypt, Hoare logic proves that each oracle preserves the two statements, and hence that they hold after every run of the adversary.

4.2 The proof of the hierarchy

Proof of Theorem 1. Fix a scheme and an adversary, and consider a run in which line 1 passes and the signature verifies, since the adversary loses every other run. We show that a forgery that is trivial at one level is trivial at the level below, so a run that the adversary wins at the lower level is one that it wins at the higher level.

- From 1 to 0. Line 1 gives $1 \leq t - |CS|$, so $\mathsf{tf}_1(M)$ gives $1 \leq |S_1(M)|$, which is $\mathsf{tf}_0(M)$.
- From 2 to 1. Let lr be a request with $lr.\mathsf{msg} = M$ and $\mathsf{tf}_2(lr)$. By the first statement of Lemma 1, $t - |CS| \leq |S_2(lr)| \leq |S_1(M)|$, which is $\mathsf{tf}_1(M)$.
- From 3 to 2. By definition, $\mathsf{tf}_3(lr)$ implies $\mathsf{tf}_2(lr)$.
- From 4 to 3. Let lr be a request with $\mathsf{tf}_4(lr)$. Then $1 \leq t - |CS| \leq |S_2(lr)|$. A request with a non-empty S_2 is in L, and hence Lemma 1 gives $S_2(lr) \subseteq S_3(lr) \subseteq S_4(lr)$. With $S_2(lr) = S_4(lr)$ this forces $S_2(lr) = S_3(lr)$, which is $\mathsf{tf}_3(lr)$.

The strong predicate tsf_i adds the same condition on SVf at every level. The two steps between strong notions therefore follow from the steps above, and tsf_i implies tf_i, which gives the last line of the theorem. □

The proof in EasyCrypt has the same structure. Nine lemmas compare the predicates on a final state that satisfies the invariants. One lemma in probabilistic relational Hoare logic turns each of them into the inequality between the games. Two runs that differ only in the notion make the same calls to the scheme and to the adversary, so this lemma couples them to end with the same state and the same result of Vf.

4.3 The game without line 11

The game without line 11 still requires the request to contain a token for i, but not one that server i issued. A server $i \in S_2(lr)$ then need not satisfy $lr.\mathsf{PP}(i) \in \mathrm{PP}_i$, so $S_2(lr) \subseteq S_3(lr)$ can fail. In the counterexample of CPZ, some server in $S_2(lr)$ is not in $S_3(lr)$ [CPZ24, Appendices D and G]. Every server of the scheme $\mathsf{RTS}_4^*[\mathsf{DS}]$ of Section 6, a variant of BTZ's reference scheme $\mathsf{RTS}_4[\mathsf{DS}]$, issues the same token ε. A server checks that the token echoed to it is ε, but not that it issued it.

Proposition 1 (The game without line 11)**.** *Let $2 \le t < ns$ and let DS be a correct signature scheme. In the game without line 11, some adversary against $\mathsf{RTS}_4^*[\mathsf{DS}]$ wins at level 3 with probability 1 and at level 4 with probability 0.*

The adversary corrupts no server, makes no query to PPO, and asks servers 1 to t to sign one request lr that names servers 1 to t and echoes ε to each. Every server answers, and the adversary outputs the signature that consists of lr, the set $\{1, \ldots, t\}$ and the t partial signatures, which verifies for $lr.\mathsf{msg}$. No token was issued, so $S_3(lr)$ is empty, and all t servers signed, so $S_2(lr) = S_4(lr) = \{1, \ldots, t\}$. At level 4 the forgery is trivial, since $S_2(lr) = S_4(lr)$, and the adversary loses. At level 3 it is not trivial, since $S_2(lr) \neq S_3(lr)$, and the adversary wins. The inequality $\mathbf{Adv}^{\text{ts-uf-3}}_{\mathsf{TS}}(\mathcal{A}) \le \mathbf{Adv}^{\text{ts-uf-4}}_{\mathsf{TS}}(\mathcal{A})$ therefore fails in the game without line 11. With line 11, Theorem 1 gives $\mathbf{Adv}^{\text{ts-uf-3}}_{\mathsf{TS}}(\mathcal{A}) \le \mathbf{Adv}^{\text{ts-uf-4}}_{\mathsf{TS}}(\mathcal{A})$ for the same scheme and adversary.

4.4 Line 11 and the server check

Theorem 1 uses line 11 and not the server check, so PS may answer any request that the oracle passes to it. Line 11 can instead be left out when the servers make the server check themselves, and no advantage changes.

For a scheme TS, let TS' be the scheme whose server i keeps, besides its state in TS, the list of the tokens that it issued. Its preprocessing runs that of TS and appends the new token to the list. Its partial signing returns $\perp$ when $lr.\mathsf{PP}(i)$ is not in the list, and otherwise runs that of TS. Its other algorithms are those of TS, so TS' makes the server check.

Theorem 2 (Line 11 as a server check)**.** *For every scheme TS, every adversary $\mathcal{A}$ and each of the eight games, the probability that $\mathcal{A}$ wins the game against TS equals the probability that $\mathcal{A}$ wins the same game without line 11 against TS'.*

Proof. We couple the two games. The list of server i of TS' always equals PP_i, since both grow by the token of each query to $\mathrm{PPO}(i)$. When line 11 refuses a query, server i of TS' refuses it too, without running TS, and when line 11 accepts it, TS' runs the partial signing of TS on the same state. The two games therefore make the same calls to TS and give the adversary the same answers. Their records agree, except that the list L of the game without line 11 can hold further requests, which TS' refused. No server signed a request that is only in this list, so its set S_2 is empty and it adds nothing to S_1. The predicates of levels 0 and 1 read S_1 alone, which therefore agrees in the two games. When CS satisfies line 1, the predicates of levels 2 to 4 are false on the further requests, so these requests make no forgery trivial. When CS violates line 1, the adversary loses both games. □

In EasyCrypt the two schemes have different types of server state. The game without line 11 is therefore stated over a second instance of the model, whose server state is a triple of the state in TS, the index of the server and the list.

5 From TS-UF-0 to TS-UF-1

BTZ also prove that TS-UF-0 implies TS-UF-1, with a loss of $\binom{ns}{t-1}$ [BTZ22, Theorem 3.1 and Appendix B], and call this loss acceptable where t is expected to be constant [BTZ22, Section 3.3].

For an adversary $\mathcal{A}$, the adversary $\mathcal{B}$ runs $\mathcal{A}$. When $\mathcal{A}$ corrupts a set CS with $|CS| < t$, the adversary $\mathcal{B}$ draws a uniform set ECS of $t - 1 - |CS|$ further servers outside CS. It corrupts $CS \cup ECS$, a set of $t - 1$ servers, which satisfies line 1. If $t \le |CS|$, it takes ECS empty, since $\mathcal{A}$ then loses by line 1. It gives $\mathcal{A}$ the keys of CS only. The game of $\mathcal{B}$ treats the servers of ECS as corrupted and refuses queries to them, so $\mathcal{B}$ answers the queries of $\mathcal{A}$ to these servers itself, and forwards every other query. For these servers, $\mathcal{B}$

keeps the states and the issued tokens, makes the checks of lines 10 and 11, and runs the algorithms of the scheme.

Theorem 3 (Loose implication, BTZ Theorem 3.1)**.** *For every scheme* TS *and every adversary* $\mathcal{A}$*, the adversary* $\mathcal{B}$ *satisfies*

$$\mathbf{Adv}^{\text{ts-uf-1}}_{\mathsf{TS}}(\mathcal{A}) \leq \binom{ns}{t-1} \cdot \mathbf{Adv}^{\text{ts-uf-0}}_{\mathsf{TS}}(\mathcal{B}).$$

As in BTZ's theorem, $\mathcal{B}$ runs in about the time of $\mathcal{A}$ and makes at most as many queries to each oracle as $\mathcal{A}$.

If $\mathcal{A}$ wins at level 1 with a forgery on M, then $|S_1(M)| \leq t - 1 - |CS|$. Since $\mathcal{B}$ answers the queries to the servers of ECS as the game would, the view of $\mathcal{A}$, and with it $S_1(M)$, does not depend on ECS. The set ECS is uniform among the sets of $t - 1 - |CS|$ servers outside CS, and $S_1(M)$ is a set of at most $t - 1 - |CS|$ servers outside CS. Hence ECS contains $S_1(M)$ with probability at least $1/\binom{ns-|CS|}{t-1-|CS|}$, which is at least $1/\binom{ns}{t-1}$. In that case the set $S_1(M)$ of the game of $\mathcal{B}$, which records only the servers outside $CS \cup ECS$, is empty, and $\mathcal{B}$ wins at level 0.

The proof in EasyCrypt has three steps. First, we draw ECS after the game at level 1 has ended, and a counting argument bounds from below the probability that $\mathcal{A}$ wins and ECS contains $S_1(M)$. Second, a program equivalence moves the draw of ECS to just after the corruption, where $\mathcal{B}$ makes it. Third, a coupling relates the resulting game to the game of $\mathcal{B}$ at level 0. In this coupling, $\mathcal{B}$ makes the same calls to the scheme for the servers in ECS, on the same states and in the same order, as the game at level 1, so the coupling holds for every scheme.

Theorem 3 is proved in EasyCrypt for every scheme, but if the scheme is not pure, its adversary $\mathcal{B}$ can use the secret keys of honest servers. For the servers in ECS, BTZ's $\mathcal{B}$ runs SPP and PS on states that it computes from their keys. In the model, $\mathcal{B}$ instead calls the procedures of the scheme, and these procedures may read the variables of its module. A pure scheme never writes these variables, so they do not depend on the keys, and $\mathcal{B}$ computes only with the keys of $CS \cup ECS$. The scheme of Section 3.4 stores all secret keys in a variable, so for this scheme $\mathcal{B}$ reads the keys of honest servers, and the TS-UF-0 security of the scheme need not bound the advantage of $\mathcal{B}$. The argument for pure schemes is on paper and not in EasyCrypt.

5.1 The loss is optimal

We show that every implication from TS-UF-0 to TS-UF-1 that holds for every scheme has a loss of at least $1/(1/\binom{ns}{t-1} + \delta)$, where δ is the advantage of an adversary against a signature scheme DS. For this we give a scheme built from DS whose TS-UF-0 advantage is at most $1/\binom{ns}{t-1}$ plus a multi-user unforgeability advantage against DS, and an adversary that breaks it at level 1 with probability 1.

5.1.1 The scheme and the level-1 adversary

The scheme CTS[DS] is built from a correct signature scheme DS, with signing algorithm DS.Sign and verification algorithm DS.Vf. Key generation draws a seed s, gives server j the signing key $\mathsf{kd}(s, j)$ of DS, and draws a uniform set T^* of $t - 1$ servers. Here kd is an arbitrary function from seeds and servers to signing keys of DS, on which we make no assumption. This includes independent key generation, where the seed is a tuple of ns independently drawn signing keys and $\mathsf{kd}(s, j)$ is the j-th of them. The verification key of CTS[DS] is T^* together with the verification keys $\mathsf{DS.Pk}(\mathsf{kd}(s, j))$ of the servers, where DS.Pk maps a signing key to its verification key. Preprocessing returns the token ε, the only token of the scheme. Server i answers every request lr with $i \in lr.\mathsf{SS}$, whatever the size of $lr.\mathsf{SS}$, by signing $lr.\mathsf{msg}$ under its own key. A signature is a map from servers to signatures of DS. Verification accepts (vk, M, sig) when the servers of sig lie in $[1..ns]$ and every signature in sig verifies on M under the key of its server. It also requires that there are at least t of these servers, or that they are exactly T^*.

The adversary $\mathcal{A}_1$ corrupts no server, reads T^* from the verification key, has every server of T^* issue a token, so that line 11 accepts its requests, and has them sign one request for a message M with signer set T^*. The map from each server of T^* to its partial signature is a signature that Vf accepts, since the set of signers is exactly T^*. The forgery is not trivial at level 1, since $|S_1(M)| = t - 1 < t - |CS|$.

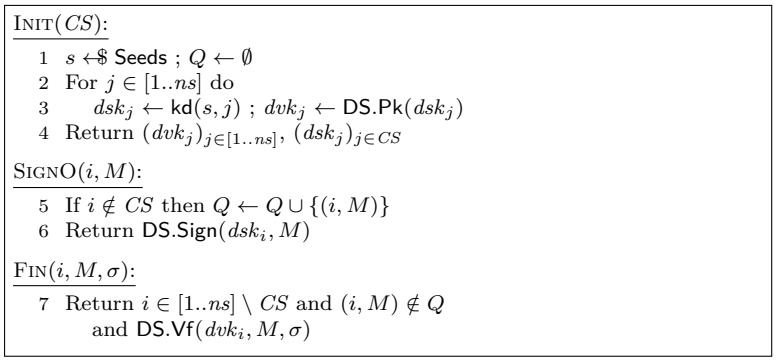

Figure 3: The game MU-CMA of multi-user unforgeability of DS under static corruptions, for the keys that CTS[DS] derives from one seed. The distribution Seeds is that of the seed in key generation.

5.1.2 The assumption

A forgery at level 0 is on a message M that no honest server signed. Unless $T^* \subseteq CS$, a signature that Vf accepts then contains a signature of DS on M under the key of an honest server, which is a forgery against DS. The adversary holds the keys of CS and obtains signatures from the other servers, so the assumption must cover many keys, some of them known to the adversary. The assumption is therefore the multi-user unforgeability of DS under static corruptions, for the keys that CTS[DS] derives from one seed. Figure 3 gives the game MU-CMA. The adversary names a set CS of servers, receives the verification keys of all servers and the signing keys of the servers in CS, and can ask any server to sign any message. It wins with a signature on a message M that verifies under the key of a server $i \notin CS$, if it never asked server i to sign M. We write $\mathbf{Adv}^{\text{mu-cma}}_{\mathsf{DS}}(\cdot) = \Pr[\mathsf{MU\text{-}CMA}(\cdot) = 1]$ for the advantage of an adversary in this game.

5.1.3 The reduction and the result

For an adversary $\mathcal{A}$ against CTS[DS] at level 0, the adversary $\mathcal{C}(\mathcal{A})$ in MU-CMA runs $\mathcal{A}$ and plays the level-0 game for it. It names the set CS that $\mathcal{A}$ corrupts, draws T^* itself, and gives $\mathcal{A}$ the verification key of CTS[DS] and the signing keys of CS. It answers the preprocessing queries of $\mathcal{A}$ to honest servers with ε and records them. It makes the checks of lines 10 and 11 for each signing query of $\mathcal{A}$, and answers a query to server i that passes them by asking its oracle to sign $lr.\mathsf{msg}$ under the key of server i. When $\mathcal{A}$ returns a forgery on M, the adversary $\mathcal{C}(\mathcal{A})$ returns a signature in it that belongs to a server outside CS and verifies on M under the key of that server, and an arbitrary output if there is none.

Theorem 4 (The loss is optimal)**.** *Let $2 \le t < ns$ and let* DS *be a correct signature scheme. Then* $\mathbf{Adv}^{\text{ts-uf-1}}_{\mathsf{CTS[DS]}}(\mathcal{A}_1) = 1$. *Moreover, every adversary $\mathcal{A}$ that reads or writes no variable of the level-0 game, of* MU-CMA *or of $\mathcal{C}$ satisfies*

$$\mathbf{Adv}^{\text{ts-uf-0}}_{\mathsf{CTS[DS]}}(\mathcal{A}) \le \frac{1}{\binom{ns}{t-1}} + \mathbf{Adv}^{\text{mu-cma}}_{\mathsf{DS}}(\mathcal{C}(\mathcal{A})).$$

The condition on $\mathcal{A}$ is the one of Section 3, that an adversary cannot read or write the variables of the games or of a reduction that runs it.

Suppose that an implication with factor c holds for CTS[DS] and $\mathcal{A}_1$, so that $1 = \mathbf{Adv}^{\text{ts-uf-1}}_{\mathsf{CTS[DS]}}(\mathcal{A}_1) \le c \cdot \mathbf{Adv}^{\text{ts-uf-0}}_{\mathsf{CTS[DS]}}(\mathcal{A})$ for the level-0 adversary $\mathcal{A}$ that it provides. Theorem 4 then gives $1/(1/\binom{ns}{t-1} + \delta) \le c$ for $\delta = \mathbf{Adv}^{\text{mu-cma}}_{\mathsf{DS}}(\mathcal{C}(\mathcal{A}))$, and this lower bound is close to $\binom{ns}{t-1}$ when δ is small.

Now let DS be multi-user unforgeable and let $\binom{ns}{t-1}$ be large. Since $\mathcal{C}(\mathcal{A})$ uses about the same resources as $\mathcal{A}$, Theorem 4 bounds the advantage at level 0 against CTS[DS] of every adversary with reasonable resources by a small number, while $\mathcal{A}_1$ wins at level 1 with probability 1. Hence CTS[DS] is TS-UF-0-secure and not TS-UF-1-secure, and the bound of Theorem 3 is useful only when $\binom{ns}{t-1}$ is small. The lower bound concerns implications that hold for every scheme, and says nothing about the loss of a reduction for a particular scheme such as BLS or FROST, which a meta-reduction [Cor02] would bound.

Proof of Theorem 4. The adversary $\mathcal{A}_1$ wins at level 1 in every run, as shown above. If $\mathcal{A}$ wins at level 0 with a forgery on M, then $S_1(M)$ is empty. Its signature contains signatures of DS on M from at least t servers, one of which is outside CS by line 1, or from exactly the servers of T^*. The adversary names CS before key generation draws T^*, which is uniform among the $\binom{ns}{t-1}$ sets of $t-1$ servers, so $T^* \subseteq CS$ with probability at most $1/\binom{ns}{t-1}$. In the other runs that $\mathcal{A}$ wins, $\mathcal{C}(\mathcal{A})$ therefore returns a signature on M under the key of an honest server i. Each signing query of $\mathcal{C}(\mathcal{A})$ for server i on a message M' answers a query of $\mathcal{A}$ after which the level-0 game adds i to $S_1(M')$. Since $S_1(M)$ is empty, $\mathcal{C}(\mathcal{A})$ never asked server i to sign M, and it wins $\mathsf{MU\text{-}CMA}$. □

We prove Theorem 4 in EasyCrypt for an abstract DS. A coupling relates the level-0 game to $\mathsf{MU\text{-}CMA}$, and its invariant states that every query of $\mathcal{C}(\mathcal{A})$ to its signing oracle is recorded in S_1.

6 The Separations

BTZ show that no implications hold beyond those of Theorems 1 and 3 [BTZ22, Appendix A]. They use three of their four reference schemes and a variant of the fourth, each secure for one notion X and broken at another notion Y, meaning that some adversary wins the game of Y against it with probability 1.

6.1 The reference schemes

The reference schemes $\mathsf{RTS}_\ell[\mathsf{DS}]$ of BTZ, for $\ell \in \{1,2,3,4\}$, are built from a standard signature scheme DS [BTZ22, Figure 14]. Each server has its own key pair of DS, and the verification key of the scheme lists the verification keys of all servers. Preprocessing returns the constant token ε, so line 11 accepts a signing query to server i only if server i has issued a token and the request echoes ε to it. A signature is a triple of a request, a set F of servers and a partial signature from each server in F. Table 1 gives the partial signing and the verification of each scheme. For $\ell \in \{2,3,4\}$, the scheme $\mathsf{RTS}_\ell[\mathsf{DS}]$ also has a strong verification algorithm, and $\mathsf{SVf}(vk, lr, sig)$ accepts when $\mathsf{Vf}(vk, lr.\mathsf{msg}, sig)$ accepts and the request inside sig equals lr.

Remark 1 (Uniqueness of SVf for $\mathsf{RTS}_2[\mathsf{DS}]$). BTZ define the strong notions only for schemes whose SVf accepts at most one signature for each pair (vk, lr) [BTZ22, Section 3.2]. In BTZ's $\mathsf{RTS}_2[\mathsf{DS}]$, verification accepts any set F of at least t servers that the request inside the signature names. Hence, for a request lr that names servers 1 to $t+1$, SVf accepts two signatures for (vk, lr), one with $F = \{1, \ldots, t\}$ and one with $F = \{2, \ldots, t+1\}$. We therefore change the verification of $\mathsf{RTS}_2[\mathsf{DS}]$ to require that F be the set of the t smallest servers that the request names. The request then determines F, and, when DS is unique, it determines the signature.

Table 1: The reference schemes $\mathsf{RTS}_\ell[\mathsf{DS}]$ of BTZ [BTZ22, Figure 14], as we mechanise them. A signature verifies for a message M if the request lr inside it has $lr.\mathsf{msg} = M$, every partial signature in it verifies, on what its server signs, under the key of that server, $t \le |F|$, and F satisfies the condition of the last column.

ℓ	What server i signs	Server i answers only if	Condition on F
1	$lr.\mathsf{msg}$	$i \in lr.\mathsf{SS}$ and $t \le \lvert lr.\mathsf{SS}\rvert$	$F \subseteq lr.\mathsf{SS}$
2	lr	$i \in lr.\mathsf{SS}$ and $t \le \lvert lr.\mathsf{SS}\rvert$	F is the set of the t smallest servers of $lr.\mathsf{SS}$ (BTZ: $F \subseteq lr.\mathsf{SS}$)
3	lr	$i \in lr.\mathsf{SS}$, $t \le \lvert lr.\mathsf{SS}\rvert$ and $lr.\mathsf{PP}(i) = \varepsilon$	$F = \{\, j \in lr.\mathsf{SS} : lr.\mathsf{PP}(j) = \varepsilon \,\}$
4	lr	$i \in lr.\mathsf{SS}$, $t \le \lvert lr.\mathsf{SS}\rvert$ and $lr.\mathsf{PP}(i) = \varepsilon$	$F = lr.\mathsf{SS}$

BTZ separate $\mathsf{TS\text{-}UF\text{-}4}$ from $\mathsf{TS\text{-}SUF\text{-}2}$ with a variant $\mathsf{RTS}^*_4[\mathsf{DS}]$ of the fourth reference scheme, built over a signature scheme that is not SUF-CMA-secure [BTZ22, Appendix A]. From a unique signature scheme DS, they build the scheme DS^+ that prepends a 0 bit to every signature of DS and whose verification ignores this bit. The scheme DS^+ is UF-CMA-secure when DS is, but it is not SUF-CMA-secure, since

flipping the first bit of a valid signature gives another valid signature. The scheme $\mathsf{RTS}_4^*[\mathsf{DS}]$ is $\mathsf{RTS}_4[\mathsf{DS}^+]$, except that its SVf also requires the first bit of every partial signature to be 0. This SVf accepts at most one signature for each pair (vk, lr), since DS is unique. The game of TS-UF-4 does not run SVf, so $\mathsf{RTS}_4^*[\mathsf{DS}]$ is TS-UF-4-secure whenever $\mathsf{RTS}_4[\mathsf{DS}^+]$ is.

In EasyCrypt, the four schemes are the modules RTS1 to RTS4 of one instance of the model, so they share one type of partial signatures, the signatures of DS^+. Honest servers set the first bit to 0. The verification of RTS2 and RTS3 requires this bit to be 0, while that of RTS1 and RTS4 ignores it. Hence RTS2 and RTS3 behave as schemes over DS, as the strong claims of Proposition 2 require, and RTS1 and RTS4 as schemes over DS^+. Strong verification requires every first bit to be 0, so the module RTS4 is $\mathsf{RTS}_4^*[\mathsf{DS}]$.

Proposition 2 (Security of the reference schemes)**.** (BTZ [BTZ22, Proposition A.1], stated without proof and not mechanised.) *Let $2 \le t < ns$. If* DS *is UF-CMA-secure, then* $\mathsf{RTS}_\ell[\mathsf{DS}]$ *is TS-UF-ℓ-secure for $\ell \in \{1, 2, 3, 4\}$. If* DS *is unique and SUF-CMA-secure, then* $\mathsf{RTS}_\ell[\mathsf{DS}]$ *is TS-SUF-ℓ-secure for $\ell \in \{2, 3, 4\}$.*

BTZ state Proposition 2 for their $\mathsf{RTS}_2[\mathsf{DS}]$, before the change of Remark 1, so the proposition says nothing about the scheme that we mechanise. A reduction to the security of DS needs only that a valid signature contains t valid partial signatures on its request, so we expect the proposition to hold for this scheme as well. For RTS1, which is over DS^+, the proposition applies since DS^+ is UF-CMA-secure when DS is.

6.2 The attacks

Theorem 5 (Separations)**.** *Let $2 \le t < ns$ and let* DS *be a correct signature scheme. In each of the following games, some adversary that corrupts no server wins with probability 1:*

- *the game of TS-UF-2 against* $\mathsf{RTS}_1[\mathsf{DS}]$*,*
- *the game of TS-UF-3 against* $\mathsf{RTS}_2[\mathsf{DS}]$*,*
- *the game of TS-UF-4 against* $\mathsf{RTS}_3[\mathsf{DS}]$*,*
- *the game of TS-SUF-2 against* $\mathsf{RTS}_4^*[\mathsf{DS}]$*.*

Table 2: The separations of BTZ [BTZ22, Appendix A]. By Theorem 5, each scheme is broken at the notion of the third column, and by Proposition 2 it is secure for the notion of the second, for $\mathsf{RTS}_2[\mathsf{DS}]$ only if the proposition holds for the scheme of Remark 1.

Scheme	Secure for	Broken at
$\mathsf{RTS}_1[\mathsf{DS}]$	TS-UF-1	TS-UF-2
$\mathsf{RTS}_2[\mathsf{DS}]$	TS-SUF-2	TS-UF-3
$\mathsf{RTS}_3[\mathsf{DS}]$	TS-SUF-3	TS-UF-4
$\mathsf{RTS}_4^*[\mathsf{DS}]$	TS-UF-4	TS-SUF-2

In Table 2, the first clause of Proposition 2 gives the second column for $\mathsf{RTS}_1[\mathsf{DS}]$ and $\mathsf{RTS}_4^*[\mathsf{DS}]$, and the second clause gives it for $\mathsf{RTS}_3[\mathsf{DS}]$. For $\mathsf{RTS}_2[\mathsf{DS}]$, the second column needs Proposition 2 for the scheme of Remark 1, which we expect but do not prove. By Theorem 1, a scheme that is secure for a notion X is secure for every notion that X implies, and a scheme broken at a notion Y is broken at every notion that implies Y. Each scheme therefore separates every notion for which it is secure from every notion at which it is broken:

- the scheme $\mathsf{RTS}_1[\mathsf{DS}]$ separates TS-UF-0 and TS-UF-1 from the six other notions,
- the scheme $\mathsf{RTS}_2[\mathsf{DS}]$ separates TS-UF-0 to TS-UF-2 and TS-SUF-2 from the four other notions,
- the scheme $\mathsf{RTS}_3[\mathsf{DS}]$ separates the six notions other than TS-UF-4 and TS-SUF-4 from these two,
- the scheme $\mathsf{RTS}_4^*[\mathsf{DS}]$ separates TS-UF-0 to TS-UF-4 from the three strong notions.

These four items give a separation for each of the thirty pairs of notions X and Y with no path from X to Y along the solid and dashed arrows of Figure 1. We count the dashed arrow as an implication, as BTZ do, although the scheme $\mathsf{CTS}[\mathsf{DS}]$ of Section 5.1 separates TS-UF-0 from TS-UF-1 when $\binom{ns}{t-1}$ is large and DS is multi-user unforgeable. These separations depend on Proposition 2, which we do not mechanise.

Each attack first has servers 1 to $t+1$ issue a token, so that line 11 accepts its signing queries, while BTZ's attacks make no such queries.

- Against $\mathsf{RTS}_1[\mathsf{DS}]$, the adversary asks servers 1 to $t-1$ to sign a message M under a request that names servers 1 to t, and server t to sign M under a second request that names servers 2 to $t+1$. The partial signatures are on M alone, so together they form a valid signature with the first request. Neither request has t partial signatures, so the forgery is not trivial at level 2.
- Against $\mathsf{RTS}_2[\mathsf{DS}]$, the request names servers 1 to $t+1$ with the tokens that they issued, and only servers 1 to t sign. These are the t smallest servers of the request, so their partial signatures form a valid signature. Server $t+1$ is in $S_3(lr)$ but not in $S_2(lr)$, so the forgery is not trivial at level 3.
- Against $\mathsf{RTS}_3[\mathsf{DS}]$, the request names servers 1 to $t+1$ but echoes a token other than ε to server $t+1$, and servers 1 to t sign. Verification then requires $F = \{1, \ldots, t\}$, so their partial signatures form a valid signature. Server $t+1$ is in $S_4(lr)$ but not in $S_2(lr)$, so the forgery is not trivial at level 4.
- Against $\mathsf{RTS}^*_4[\mathsf{DS}]$, the request names servers 1 to t with the tokens that they issued, and all of them sign. The adversary flips the first bit of every partial signature. The resulting signature verifies, since Vf ignores the first bits, but SVf rejects it, so the forgery is not trivial for TS-SUF-2.

Remark 2 (The attack against $\mathsf{RTS}_2[\mathsf{DS}]$). BTZ's attack against $\mathsf{RTS}_2[\mathsf{DS}]$ echoes the token 0 to each server that it names [BTZ22, Appendix A]. The servers of $\mathsf{RTS}_2[\mathsf{DS}]$ issue only ε, so line 11 refuses every signing query of this attack, and the attack does not win in the revised game. The attack above uses only issued tokens, and it names server $t+1$, so it needs $t < ns$, as BTZ assume for their reference schemes.

Each attack makes a fixed sequence of oracle queries. In EasyCrypt, a proof in Hoare logic shows that, in every run, its signature verifies and its forgery is not trivial, and a proof of termination then gives probability 1. These proofs use only the correctness of DS. With Theorem 1, we also prove that each scheme is broken at every notion that implies the notion of its attack. Proposition 1 is proved in the same way.

Remark 3 (The reference schemes and the server check). Proposition 2 states that $\mathsf{RTS}_3[\mathsf{DS}]$ is TS-SUF-3-secure, and the separation of TS-SUF-2 from TS-UF-3 attacks $\mathsf{RTS}_2[\mathsf{DS}]$ at level 3. These two statements are about level 3, which the revision defines only for schemes whose servers make the server check [BTZ22, Section 3.2], and the servers of $\mathsf{RTS}_2[\mathsf{DS}]$ and $\mathsf{RTS}_3[\mathsf{DS}]$ do not make it. The variants $\mathsf{RTS}_\ell[\mathsf{DS}]'$ of Theorem 2 make it, and by that theorem every adversary has the same advantage against $\mathsf{RTS}_\ell[\mathsf{DS}]'$ in the game without line 11 as against $\mathsf{RTS}_\ell[\mathsf{DS}]$ in the game with it. Hence the adversaries of Theorem 5 also win with probability 1 against $\mathsf{RTS}_\ell[\mathsf{DS}]'$ in the game without line 11, and a proof of Proposition 2 for $\mathsf{RTS}_\ell[\mathsf{DS}]$ would also prove it for $\mathsf{RTS}_\ell[\mathsf{DS}]'$ in that game. In EasyCrypt these attacks are corollaries of Theorem 2. With line 11, the list of server i of $\mathsf{RTS}_\ell[\mathsf{DS}]'$ equals PP_i, so its check passes on every request that line 11 accepts, and $\mathsf{RTS}_\ell[\mathsf{DS}]'$ answers as $\mathsf{RTS}_\ell[\mathsf{DS}]$. The same conclusions therefore hold in the game with line 11, which is BTZ's game for these schemes, and we argue this on paper.

Acknowledgements

We thank the EasyCrypt development team for the continued support and development of the tool. This work is funded by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the support UID/50014/2025 (https://doi.org/10.54499/UID/50014/2025), and J. D. Duarte is supported by FCT under PhD grant https://doi.org/10.54499/2024.04803.BD.

Use of AI

The author used Claude (Anthropic) to edit and proofread the text of this note, and for much of the tactic work in the proofs. The files `Subsets.ec`, of lemmas on uniform subsets, and `TSNoLine11.ec` were

written by it in full. EasyCrypt checks every proof script. The author is responsible for the result.

References

[BCK+22] Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 517–550, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-031-15985-5_18.

[BDL+26] Manuel Barbosa, François Dupressoir, Cameron Low, Charlotte Mylog, Sabine Oechsner, and Théo Vignon. Principles for pRHL proofs of cryptographic protocols: How to convince EasyCrypt that your protocol is secure. Cryptology ePrint Archive, Report 2026/1334, 2026. URL: https://eprint.iacr.org/2026/1334.

[BGHZ11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90, Santa Barbara, CA, USA, August 14–18, 2011. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-22792-9_5.

[BTZ22] Mihir Bellare, Stefano Tessaro, and Chenzhi Zhu. Stronger security for non-interactive threshold signatures: BLS and FROST. Cryptology ePrint Archive, Report 2022/833, 2022. URL: https://eprint.iacr.org/2022/833.

[Cor02] Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 272–287, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-46035-7_18.

[CPZ24] Cas Cremers, Aleksi Peltonen, and Mang Zhao. An extended hierarchy of security notions for threshold signature schemes and automated analysis of protocols that use them. Cryptology ePrint Archive, Report 2024/1920, 2024. URL: https://eprint.iacr.org/2024/1920.

[KG20] Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *SAC 2020: 27th Annual International Workshop on Selected Areas in Cryptography*, volume 12804 of *Lecture Notes in Computer Science*, pages 34–65, Halifax, NS, Canada (Virtual Event), October 21-23, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-81652-0_2.

[NTW22] Ngoc Khanh Nguyen, Eftychios Theodorakis, and Bogdan Warinschi. Lifting standard model reductions to common setup assumptions. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022: 25th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 13178 of *Lecture Notes in Computer Science*, pages 130–160, Virtual Event, March 8–11, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-030-97131-1_5.

A The Definitions in EasyCrypt

The listings below are taken verbatim from the source files. In them, `x <- e` assigns, `x <$ d` samples from a distribution, and `x <@ M.p(a)` calls a procedure. A module type lists the procedures that a module provides, and the braces after a procedure list the oracles that it may call. Operators are total functions of the logic, `x.`i` is the $i$-th component of a tuple or record, and `m.[x]` is the value of the map `m` at `x`. For an optional value `o`, `oget o` is its content and `odflt d o` is its content or the default `d`.

Besides these listings, the statements depend on the rest of `TSDefs.ec` and `TSGame.ec`, on the trivial-forgery predicates of `TSSets.ec` and on the state record of `TSState.ec`. The statements of Proposition 1 and Theorems 2 to 5 depend in addition on the schemes, adversaries and reductions that they name.

The scheme. A leader request is a triple of a message, a set of servers and a finite map of tokens, so `lr.`1`, `lr.`2` and `lr.`3` are lr.msg, lr.SS and lr.PP.

TSDefs.ec

```
module type TS  = {
  proc keygen() : public_key * public_keys * secret_keys
  proc signer_preproc(st : server_state) : preproc_token * server_state
  proc leader_preproc(i : int, pp: preproc_token) : unit
  proc signer_partialsig(st : server_state, lr : leader_request)
    : partialsig option * server_state
  proc leader_sigreq(m: message, ss : int fset, pp: preproc_token fset)
    : leader_request
  proc leader_aggregate(lr: leader_request, ps: partialsig option fset) : sig
  proc verif(vk: public_key, M: message, sign: sig) : bool
  proc sverif(vk: public_key, lr: leader_request, sign: sig) : bool
}.
```

The adversary. The adversary is parametrised by the two oracles.

TSDefs.ec

```
module type PPO = {
  proc preproc(i : int) : preproc_token option
}.

module type PSignO = {
  proc psign(i: int, lr: leader_request) : partialsig option
}.

(* The adversary sees only the two oracles.  The scheme module may be
   stateful and probabilistic, so handing it to the adversary would let it
   run the honest signers' code on their state outside the oracles,
   bypassing the game's bookkeeping.  Public
   algorithms (Vf, leader steps) need no secrets, so the adversary can run
   its own copy of them. *)
module type TS_UFAdversary(P: PPO, S: PSignO) = {
  proc corrupt() : int fset { }
  proc run(vk : public_key, aux : public_keys, sks : secret_keys) : message * sig
}.
```

The signing oracle. The game keeps HS, PP_i, L, S_1 and S_2 in the fields `hs`, `pp`, `L`, `s1` and `s2` of the record `_state`, and the states st_i of the honest servers in the map `_sst`. The module `T` is the scheme, the set `all_players` is $[1..ns]$, and `` `&` `` is the intersection of two sets. The first two conjuncts of the condition are line 10, the third requires lr to contain a token for i, and the fourth is line 11.

TSGame.ec

```
  module PSign0 = {
    proc psign(i: int, lr: leader_request) : partialsig option = {
      var psig : partialsig option;
      var st   : server_state;
      psig <- None;
      if (lr.`2 \subset all_players      /\
          i \in _state.`hs `&` lr.`2     /\
          i \in lr.`3                    /\
          oget lr.`3.[i] \in odflt [] _state.`pp.[i]) {
        (psig, st) <@ T.signer_partialsig(oget _sst.[i], lr);
        _sst.[i]   <- st;
        _state <- update_psign _state i lr psig;
      }
      return psig;
    }
  }
```

The winning predicate. The game ends with the procedure `Finish.finish`, which computes S_3 and S_4, runs Vf and returns the winning predicate. The operator `finish` below is the same condition, as a function of the notion, CS, the final state, the forgery and the bit that Vf returned. The operators `build_s3` and `build_s4` compute S_3 and S_4 as in line 18, and the operator `sverif` is SVf, not the procedure of the same name in the type of a scheme. The operators `tf0` to `tsf4` are the trivial-forgery predicates of Figure 2, and `exists_tf` is the quantifier of lines 21 and 22, over L.

TSGame.ec

```
op finish (game  : int)
          (cs    : int fset)
          (state : tsuf_state)
          (m     : message)
          (sign  : sig)
          (gv    : bool) : bool =
  let s3  = build_s3 empty state.`L state.`pp state.`hs in
  let s4  = build_s4 empty state.`L state.`hs           in
  let svf = fun lr => sverif state.`vk lr sign          in
  (* Indices 0..4 are TS-UF-0..4; 5,6,7 are TS-SUF-2,3,4 (the paper
     defines strong unforgeability only for levels 2,3,4). *)
  let triv =
    if      game = 0 then tf0 state.`s1 m
    else if game = 1 then tf1 state.`s1 cs m
    else if game = 2 then exists_tf state.`L m (tf2 state.`s2 cs)
    else if game = 3 then exists_tf state.`L m (tf3 state.`s2 s3 cs)
    else if game = 4 then exists_tf state.`L m (tf4 state.`s2 s4 cs)
    else if game = 5 then exists_tf state.`L m (tsf2 state.`s2 cs svf)
    else if game = 6 then exists_tf state.`L m (tsf3 state.`s2 s3 cs svf)
    else if game = 7 then exists_tf state.`L m (tsf4 state.`s2 s4 cs svf)
    else false
  in
  let cs_invalid = !(cs \subset all_players /\ card cs < t)   in
  gv /\ !triv /\ !cs_invalid.
```

The multi-user game. The module `MUCMA` is the game MU-CMA of Figure 3 and defines the advantage $\mathbf{Adv}^{\text{mu-cma}}_{\mathsf{DS}}(\cdot)$ of Theorem 4. The operators `dsign` and `dverif` sign and verify with DS, `dpk` maps a signing key to its verification key, and `dseed` is the distribution of the seed. The operator `offsetmap f F` is the map on F with value `f i` at i, and `FMap.filter` restricts a map to the keys that satisfy a predicate.

TSTight.ec

```
module MUCMA (B : MUAdv) = {
  var cs : int fset
  var qs : (int * msg) list       (* the queries on honest keys *)
  var ks : (int, dsk) fmap

  module O : SignO = {
    proc sign(i : int, m : msg) : dsig = {
      if (!(i \in cs)) {
        qs <- qs ++ [(i, m)];
      }
      return dsign (oget ks.[i]) m;
    }
  }

  proc main() : bool = {
    var s, i, m, sg;
    cs <@ B(O).corrupt();
    s  <$ dseed;
    ks <- offsetmap (fun j => kd s j) all_players;
    qs <- [];
    (i, m, sg) <@ B(O).forge(offsetmap (fun j => dpk (kd s j)) all_players,
                             FMap.filter (fun j (_ : dsk) => j \in cs) ks);
    return i \in all_players /\ !(i \in cs) /\ !((i, m) \in qs) /\
           dverif (dpk (oget ks.[i])) m sg;
  }
}.
```

B The Lemmas and the Assumptions

The development has no admitted proof, and we checked it with EasyCrypt release r2026.10. Besides the constraint $1 \leq t \leq ns$ on the parameters, the only axiom of the model is `sverif_unique`, the uniqueness of SVf, which no proof uses. The files `TSSep.ec` and `TSTight.ec` instantiate the model with $2 \leq t < ns$. Each of them assumes that its own DS is correct and unique, and that the distribution of the key-generation seed has total mass 1. Both files use the uniqueness of DS only to prove the uniqueness of their SVf, which discharges `sverif_unique`, so Proposition 1 and Theorems 4 and 5 use only the correctness of DS. The unforgeability of DS is not an axiom, since Theorem 4 states it as an advantage. The numbered results are the following lemmas.

- Theorem 1: `main_0_le_1`, `main_1_le_2`, `main_2_le_3`, `main_3_le_4`, `main_suf2_le_suf3`, `main_suf3_le_suf4`, `main_uf2_le_suf2`, `main_uf3_le_suf3` and `main_uf4_le_suf4`, in `TSGame.ec`.
- Lemma 1: `run_adversary_wf`, from `ppo_wf` and `psigno_wf`, in `TSGame.ec`.
- Proposition 1: `no_line11_breaks_4_to_3`, and `with_line11_holds` for the game with line 11, in `TSNoLine11.ec`.
- Theorem 2: `transform_eq` in `TSTransform.ec`.
- Theorem 3: `ts_uf0_implies_ts_uf1` in `TSLoose.ec`.
- Theorem 4: `cts_broken_uf1`, `main0_bound_cma` and `loss_optimal_cma` in `TSTight.ec`.

- Theorem 5: `uf1_not_uf2`, `suf2_not_uf3`, `suf3_not_uf4` and `uf4_not_suf2` in `TSSep.ec`, and `rts1_broken` to `rts4_broken` for the notions that imply the one attacked.
- Remark 1: `svf_unique` in `TSSep.ec`.
- Remark 3: `uf1_not_uf2_tr`, `suf2_not_uf3_tr`, `suf3_not_uf4_tr` and `uf4_not_suf2_tr`, in `TSSep.ec`.