# Efficient Forward-Secure Threshold Public Key Encryption

Rafael Kurek(✉)

Group of IT Security and Cryptography,
University of Wuppertal, Wuppertal, Germany
kurek@uni-wuppertal.de

**Abstract.** The purpose of forward-secure threshold public key encryption schemes is to mitigate the damage of secret key exposure. We construct the first CCA forward-secure threshold public key encryption scheme based on bilinear pairings with groups of prime order that is secure against adaptive and malicious adversaries in the standard model. Our scheme is very efficient since it has a non-interactive key update and decryption procedure. Additionally, our scheme does not require a trusted dealer and has optimal resilience as well as small ciphertexts of constant size. It is the first scheme which achieves all of these and that can also be implemented on standardized elliptic curves.

## 1 Preliminaries

In a standard public key encryption scheme (PKE), once an adversary gets access to the secret key the adversary is able to decrypt all ciphertexts. There are different approaches to mitigate the damage due to secret key exposure. Two of these approaches are so-called *forward-secure* public key encryption schemes and *threshold* public key encryption schemes. Forward-secure schemes allow to evolve the secret key in regular time periods, while the public key remains fixed. Thus, every adversary with an outdated secret key cannot decrypt ciphers for time periods in the past. In a $(n, k)$-*threshold* PKE the secret key is split into $n$ shares and at least $k + 1$ shares are required to decrypt a ciphertext, whereas any subset of $k$ shares is insufficient. Due to the fact that forward security and thresholds improve security guarantees against secret key exposure in a different manner, their combination can even reinforce these guarantees. For digital signature schemes their combination was first proposed by Abdalla et al. [1] and Tzeng and Tzeng [18] and later revisited [8,16,20]. For PKE, the combination of forward-secure and threshold mechanisms to a *forward-secure threshold PKE* (fst-PKE) was proposed by Libert and Yung [15]: an adversary that wants to decrypt a ciphertext which was encrypted with a fst-PKE for time period $t$, needs not only to gain $k + 1$

R. Kurek—Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement 802823, and by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901/3).


J. K. Liu and H. Cui (Eds.): ACISP 2020, LNCS 12248, pp. 330–349, 2020.
https://doi.org/10.1007/978-3-030-55304-3_17

of the stored secret key shares but it also needs to gain these key shares before time period $t$ expires. It follows that for an adversary with restricted capacities, the combination of forward-security and threshold mechanism provides additional security. To the best of our knowledge the only existing fst-PKE scheme that is CCA forward-secure against adaptive adversaries is due to Libert and Yung.[1] It is highly efficient in terms of communication rounds: the key update procedure as well as the decryption procedure are both non-interactive. For the key update this means that there is no communication at all between the key storage hosts. Besides efficiency, this is also desirable because some of these hosts might be offline or temporarily unavailable. In the case of an interactive key update their unavailability could halt the update procedure entirely or exclude these hosts from further decryption procedures because of outdated key shares. For decryption, the non-interactive procedure means that if a decryption is requested each host can either deny the decryption or provide a valid decryption share without interacting with the other hosts.

Furthermore, the scheme by Libert and Yung is robust against malicious adversaries, which means that invalid decryption shares can be detected and do not manipulate the decrypted message. Their scheme requires bilinear pairings with groups of composite order. To guarantee security, these groups must be very large which results in very expensive computation and much larger keys as well as ciphers when compared to bilinear pairings with groups of prime order. Furthermore the scheme requires a trusted dealer that delivers the initial secret key shares to all participating parties. The only fst-PKE based on pairings with groups of prime order is due to Zhang et al. [21]. It is proven CCA forward secure against adaptive adversaries in the Random Oracle Model and requires a trusted dealer. It requires $T$ elements in the public key, where $T$ is the maximum number of time periods. Moreover, the public key needs to be stored for updating the secret key, which leads to secret keys of size $T$ as well. Although the prime order groups enable fast computation the big key sizes restrict the usage for many applications.

*Our Contribution.* We present a highly efficient *forward-secure threshold* public key encryption scheme based on bilinear pairings, which can be implemented on *standardized pairing-friendly curves* with groups of prime order. This scheme provides a *non-interactive key update and decryption procedure* and requires *no trusted dealer*. In addition, this scheme provides *ciphertexts of constant size*: one bit string of length $\log T$, where $T$ is the maximum number of supported time periods, and three group elements from prime order groups. The public key is of size $\log T$ and the secret keys have size at most $\log^2 T$. The scheme has *optimal resilience*, i.e., it can tolerate $(n-1)/2$ maliciously compromised parties and is proved CCA forward secure against adaptive adversaries in the *standard model*. Furthermore, it is possible to add pro-active security to our scheme. This enables security against mobile adversaries, i.e. against adversaries which can switch between the parties they corrupt. We discuss this concept and our techniques in Sect. 4.

[1] Adaptive adversaries can corrupt parties at any time. Static adversaries need to corrupt the parties before the protocol execution begins. For a proper overview see [1].

## 2 Forward-Secure Threshold Public Key Encryption (fst-PKE)

**Definition 1.** *We adapt the definition of fst-PKE and its security from Libert and Yung [15]. A* ***Forward-secure threshold public key encryption scheme.*** *(fst-PKE)* $(T, n, k)$*-*$\Pi_{fst}$ *is defined via the following components:*
$\boldsymbol{FST.KeyGen}(n, k, T) \rightarrow (pk, (pk_i)_{i\in[n]}, (sk_{0,i})_{i\in[n]})$*. On input the maximum number of time periods* $T$*, the maximum number of parties* $n$*, and a threshold* $k$*, it outputs a common public key* $pk$*, user public keys* $(pk_i)_{i\in[n]}$*, and initial secret key shares,* $(sk_{0,i})_{i\in[n]}$*.*
$\boldsymbol{FST.KeyUpdate}(sk_{t,i}) \rightarrow sk_{t+1,i}$*. On input a secret key share for a time period* $t < T-1$*, it outputs a secret key share for the next time period* $t+1$ *and deletes the input from its storage. Else it outputs* $\perp$*.*
$\boldsymbol{FST.Enc}(t, pk, M) \rightarrow C$*. On input a time period* $t$*, a common public key* $pk$*, and a message* $M$*, it outputs a ciphertext* $C$*.*
$\boldsymbol{FST.Dec}(t, C) \rightarrow M$*. If run by at least* $k+1$ *honest and uncompromised parties on input a time period* $t$ *and a ciphertext* $C$*, it outputs a message* $M$*.*

The key generation procedure can be either a protocol between all parties or executed by a trusted dealer. The key update procedure is assumed to be non-interactive. The decryption procedure is a protocol and contains of various steps: ciphertext-verify, share-decrypt, share-verify, and combine. For simplicity we defined the input only as a time period and a ciphertext and omit the key material held by all participating parties.

**Definition 2.** ***Correctness.*** *Let* $(pk, (pk_{0,i})_{i\in[n]}, (sk_{0,i})_{i\in[n]}) \leftarrow \boldsymbol{FST.KeyGen}$ *and* $(sk_{t,i}) \leftarrow \boldsymbol{FST.KeyUpdate}(sk_{j-1,i})$ *for* $j = 1, \ldots, t$ *and* $i \in [n]$*. We call* $\Pi_{fst}$ *correct if for all messages* $M$*, all time periods* $t \in \{0, \ldots, T-1\}$*, and all subsets* $\mathcal{U} \subseteq \{sk_{t,1}, \ldots sk_{t,n}\}$ *of size at least* $k+1$ *held by uncorrupted parties it holds that*

$$\Pr[\boldsymbol{FST.Dec}(t, \boldsymbol{FST.Enc}(t, pk, M)) = M] = 1.$$

*Note that the secret keys are an implicit input to* $\boldsymbol{FST.Dec}$*.*

We adapt the robustness notion of threshold signature schemes by Gennaro et al.[10] to forward-secure threshold public key encryption schemes.

**Definition 3.** ***Robustness.*** *A forward-secure threshold PKE* $\Pi_{fst}$ *is* $(n, k_1, k_2)$*-robust if in a group of* $n$ *parties, even in the presence of an adversary who halts up to* $k_1$ *and corrupts maliciously* $k_2$ *parties,* $\boldsymbol{FST.Keygen}$ *and* $\boldsymbol{FST.Dec}$ *complete successfully.*

Note that malicious adversaries can either deviate from the protocol in any way and especially halt some parties. Hence, they are stronger than halting adversaries, see [1].

**CCA Forward Security.** Chosen ciphertext attack (CCA) forward security against adaptive (static)[1] adversaries is defined by the following game between a challenger and an adversary $\mathcal{A}$. Let $\mathcal{B}$ and $\mathcal{G}$ be the sets of indices, which denote the corrupted and uncorrupted parties, respectively. Initially $\mathcal{B}$ is empty and $\mathcal{G} = \{1, \ldots, n\}$. The challenger (on behalf of the uncorrupted parties) and the adversary (on behalf of the corrupted parties) run **FST.KeyGen**$(n, k, T)$. The adversary receives the common public key $pk$, all user public keys $(pk_i)_{i\in[n]}$ and the initial user secret key shares $(sk_{i,0})_{i\in\mathcal{B}}$. The adversary has access to the following oracles:

**Break-In***(t',j).* On input time period $t'$ and index $j \in \mathcal{G}$, the challenger checks if $|\mathcal{B}| < k$. If this holds, the challenger removes $j$ out of $\mathcal{G}$ and adds it to $\mathcal{B}$. If $sk_{t,j}$ is already defined, i.e. after **FST.KeyGen** had finished, it is delivered to $\mathcal{A}$. If $|\mathcal{B}| = k$, the challenger outputs $sk_{t,j}$ for all $j \in \mathcal{G}$.[2]
**Challenge**$(t^*, M_0, M_1)$. The adversary submits a time period $t^*$ and two messages $M_0, M_1$. The challenger picks a bit $b$ uniformly at random and responds with a challenge ciphertext $C^* =$**FST.Enc**$(t^*, pk, m_b)$.
**Dec***(t,C).* On input time period $t$ and ciphertext $C$, the challenger (on behalf of the uncorrupted parties) and the adversary $\mathcal{A}$ (on behalf of the corrupted parties) run the decryption protocol **FST.Dec**$(t, C)$. The output of this execution is delivered to $\mathcal{A}$. If **Challenge**$(t^*, M_0, M_1)$ has already been queried and $C^*$ is the response to this query then query **Dec**$(t^*, C^*)$ is disallowed.
**Guess***(b').* The adversary outputs its guess $b' \in \{0, 1\}$. The challenger outputs 1 if $b = b'$, else 0. The game stops.

The adversary is allowed to make $k + 1$ queries **Break-In**$(t', j)$ one query **Challenge**$(t^*, m_0, m_1)$, and multiple queries **Dec**$(t, C)$, in any order, subject to $0 \leq t^* < t'_{k+1} < T$, where $t'_{k+1}$ is the time period of the $k + 1$-th query to **Break-In**. After **Break-In**$(t'_{k+1}, j)$, **Dec**$(t, C)$ cannot be queried anymore. **Guess**$(b')$ can only be queried after **Challenge**$(t^*, M_0, M_1)$. For all queries the time periods must be in $[0, \ldots, T - 1]$.

**Definition 4.** *Let $\mathcal{A}$ be an adaptive (static) adversary playing the CCA forward-security game for a fst-PKE $(T, n, k)$-$\Pi_{fst}$. It $(t_{\mathcal{A}}, \varepsilon_{\mathcal{A}})$-breaks the CCA forward security of $(T, n, k)$-$\Pi_{fst}$, if it runs in time $t_{\mathcal{A}}$ and*

$$|\Pr[\boldsymbol{Guess}(b') = 1] - 1/2| \geq \varepsilon_{\mathcal{A}}.$$

The only difference between the CPA and CCA security game is that the adversary has no access to the decryption oracle in the former game.

[1] In the static security model the adversary has to submit its choice of $k$ parties it wants to corrupt before receiving the public key. In the adaptive model it can corrupt the parties at any time. For a proper overview see [1].

[2] Note that this case can only occur for time periods $t > 0$, i.e. after **FST.KeyGen** had finished. Otherwise the adversary would have no possibility to win the security game.

**Definition 5.** *Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order $q$ with generators $g_1, g_2, g_T$. We call $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ a bilinear pairing if: i. $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ for all $a, b \in \mathbb{Z}_q$, ii. $e(g_1, g_2) \neq 1_T$, and iii. $e$ can be efficiently computed. If there is no efficiently computable isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$ we call it a Type-3 pairing. For more information we refer to [5, 7].*

The following definition of hierarchical identity-based encryption schemes (HIBE)is reproduced from [6].

**Definition 6.** *A hierarchical identity-based encryption scheme (HIBE) $\Pi_{HIBE}$ is defined via the following algorithms:* $\textbf{\textit{HIBE.Setup}}(\ell) \rightarrow (pk, sk_0)$, $\textbf{\textit{HIBE.KeyDerive}}(id, sk_{id'}) \rightarrow sk_{id'}$, *where id is a prefix of id'*, $\textbf{\textit{HIBE.Enc}}(id, pk, M) \rightarrow C$. $\textbf{\textit{HIBE.Dec}}(id, sk_{id}, C) \rightarrow M$.

*The CPA (CCA) security can be defined analogously to forward-security, see [6].*

**Definition 7.** *A digital signature scheme $\Sigma$ is defined via the following algorithms:* $\textbf{\textit{Sig.Keygen}} \rightarrow (vk, signk)$, $\textbf{\textit{Sig.Sign}}(signk, M) \rightarrow \sigma$, $\textbf{\textit{Sig.Verify}}(vk, M, \sigma) \rightarrow b$, *where* $b \in \{0, 1\}$.

**Strong Existential Unforgeability Under a One Chosen Message Attack (sEUF-1CMA).** In the sEUF-1CMA security game the adversary is allowed to query one signature and has to forge a signature for any message. The only restriction is that it cannot output the same pair of message *and* signature as for the query. If the message is the same then the signature must differ.

## 3 Our CCA Forward-Secure Threshold PKE

The key generation phase our fst-PKE uses the distributed key generation protocol **DKG** by Gennaro et al. [10]. This protocol is instantiated with the group $\mathbb{G}_2$ from the bilinear pairing we use in our fst-PKE. It outputs a common public key $pk = g_2^x \in \mathbb{G}_2$ as well as user public keys $pk_i = g_2^{x_i} \in \mathbb{G}_2$ for all parties $P_i, i \in [n]$. The user public keys are required to check the decryption shares for validity and hence provide robustness against malicious adversaries in our fst-PKE. Moreover, this protocol provides to each party $P_i$ a secret share $h^{x_i} \in \mathbb{G}_1$ of $h^x \in \mathbb{G}_1$.[3] We apply the secret shares to our fst-PKE by updating them to the first time period. We want to emphasize that for forward security it is crucial to erase the plain values $x_i, h^{x_i}$ for all $i \in [n]$ from every storage. Our fst-PKE scheme $(T, n, k)$-$\Pi_{fst}$ is defined as follows.

**Common Parameters.** The common parameters consist of a the description of a cryptographic Type-3 pairing group with groups of order $q$, the description

[3] Note that in [10] the secret value is set as $x$ instead of $h^x$. This modification happens only internally and has no impact on the adversary's view or security.

of a cryptographic hash function $H : \{0,1\}^* \to \mathbb{Z}_q$, a value $\ell$ s.t. $T = 2^\ell$ is the number of time periods, random group elements $h, h_0, \ldots, h_{\ell+1} \leftarrow \mathbb{G}_1$, and random generators $g_2, \tilde{h} \leftarrow \mathbb{G}_2$.

**FST.KeyGen**. The $n$ parties run the $\mathbf{DKG}(n,k)$ protocol from Fig. 1. Subsequently, each party $P_i$ holds the common public key $pk = g_2^x \in \mathbb{G}_2$, all user public keys $(g_2^{x_j})_{j\in[n]}$ as well as its own secret share $h^{x_i} \in \mathbb{G}_1$. Each party computes its initial secret key share $sk_{0,i}$ as

$$\left(g_2^{r_i}, h^{x_i} h_0^{r_i}, h_1^{r_i}, \ldots, h_{\ell+1}^{r_i}\right) \in \mathbb{G}_2 \times \mathbb{G}_1^{\ell+2},$$

where $r_i \leftarrow \mathbb{Z}_q$ is picked uniformly at random. The value $h^{x_i}$ is erased from the storage. The common public key $pk$ is published.[4]

**FST.KeyUpdate**$(sk_{t,i})$. We assume the $T$ time periods $0, \ldots, 2^\ell - 1$ as being organized as leaves of a binary tree of depth $\ell$ and sorted in increasing order from left to right. This means, $00\ldots0$ is the first and $11\ldots1$ is the last time period. The path from the root of the tree to a leaf node $t$ equals the bit representation $t_1 \ldots t_\ell$, where we take the left branch for $t_z = 0$ and the right one for $t_z = 1$. Prefixes of time periods correspond to internal nodes $\omega = \omega_1, \ldots, \omega_s$, where $s < \ell$. Let $r_i' \leftarrow \mathbb{Z}_q$ be picked uniformly at random. Then, we associate to each party $P_i, i \in [n]$ and each node $\omega$ a secret key:

$$(c_i, d_i, e_{i,s+1}, \ldots, e_{i,\ell+1}) = \left(g_2^{r_i'}, h^{x_i}(h_0 \prod_{v=1}^{s} h_v^{\omega_v})^{r_i'}, h_{s+1}^{r_i'}, \ldots, h_{\ell+1}^{r_i'}\right). \quad (3)$$

Given such a secret key, we derive a secret key for a descendant node $\omega' = \omega_1 \ldots \omega_{s'}$, where $s' > s$ as

$$\begin{aligned}&(c_i', d_i', e_{i,s'+1}', \ldots, e_{i,\ell+1}')\\ &= \left(c_i \cdot g_2^{r_i''}, d_i \cdot \prod_{v=s+1}^{s'} e_{i,v}^{w_v} (h_0 \prod_{v=1}^{s'} h_v^{w_v})^{r_i''}, e_{i,s'+1} \cdot h_{s'+1}^{r_i''}, \ldots, e_{i,\ell+1} \cdot h_{\ell+1}^{r_i''}\right),\end{aligned}$$

where $r_i'' \leftarrow \mathbb{Z}_q$ is picked uniformly at random.

We define $C_t$ as the smallest subset of nodes that contains an ancestor or leaf for each time period $t, \ldots, T-1$, but no nodes of ancestors or leafs for time periods $0, \ldots, t-1$. For time period $t$, we define the secret key $sk_{i,t}$ of party $P_i$ as the set of secret keys associated to all nodes in $C_t$. To update the secret key

[4] Note that the secret share $x_i$ is computed commonly by all parties and the randomness $r_i$ is computed locally by party $P_i$ and is not a share of another random value. This approach is more efficient than computing random values commonly, especially to different bases.

**Protocol DKG**$(n, k)$**:**

**Generation of shared secret** $x$**:**

1. (a) Each Party $P_i, i \in [n]$ picks two random polynomials $a_i(z)$ and $b_i(z)$ over $\mathbb{Z}_q$ of degree $k$:

$$a_i(z) = a_{i0} + a_{i1}z + \cdots + a_{ik}z^k \text{ and}$$
$$b_i(z) = b_{i0} + b_{i1}z + \cdots + b_{ik}z^k.$$

   (b) Each party $P_i$ computes and broadcasts $C_{is} = g_2^{a_{is}}\tilde{h}^{b_{is}} \in \mathbb{G}_2$ for $s = 0, \ldots, k$.

   (c) Each party $P_i$ computes $s_{ij} = a_i(j)$ and $s'_{ij} = b_i(j) \mod q$ for $j = 1, \ldots, n$. It sends $s_{ij}, s'_{ij}$ secretly to $P_j$.

   (d) For $i = 1, \ldots, n$ each party $P_j$ checks if

$$g_2^{s_{ij}}\tilde{h}^{s'_{ij}} = \prod_{s=0}^{k}(C_{is})^{j^s}. \tag{1}$$

   If there is an index $i \in [n]$ such that the check fails, $P_j$ broadcasts a *complaint* against $P_i$.

   (e) If a dealer $P_i$ receives a complaint from $P_j$ then it broadcasts the values $s_{ij}$ and $s'_{ij}$ satisfying Equation 1.

   (f) Each party disqualifies any player that either received more than $k$ *complaints* or answered to a complaint with values that does not satisfy Equation 1.

2. Each party $P_i$ defines the set *QUAL*, which indicates all non-disqualified parties.
3. The shared secret is defined as $h^x = h^{\sum_{i\in QUAL} a_{i0}} \in \mathbb{G}_1$. Each party $P_i$ sets its share of this secret as $h^{x_i} = h^{\sum_{j\in QUAL} s_{ij}} \in \mathbb{G}_1$.

**Extracting** $y := g_2^x \in \mathbb{G}_2$**:**

4. (a) Each party $P_i, i \in QUAL$ computes and broadcasts $A_{is} = g_2^{a_{is}} \in \mathbb{G}_2$ for all $s = 0, \ldots, k$.

   (b) For each $i \in QUAL$, each party $P_j$ checks if

$$g_2^{s_{ij}} = \prod_{s=0}^{k}(A_{is})^{j^s}. \tag{2}$$

   If there is an index $i \in QUAL$ such that the check fails, $P_j$ *complaints* about $P_i$ by broadcasting $s_{ij}$ and $s'_{ij}$ that satisfy Eq. 1 but not Eq. 2.

   (c) For all parties $P_i$ who received at least one valid complaint in the extraction phase, the other parties run a reconstruction of $a_i(z)$ and $A_{is}$ for $s = 0, \ldots, k$ in the clear, using the values $s_{ij}$.

   (d) Each party $P_i$ computes the common public key as $y = \prod_{i\in QUAL} A_{i0} \in \mathbb{G}_2$ and the user public keys $pk_j$ as $g_2^{x_j} = \prod_{i\in QUAL}\prod_{k=0}^{t}(A_{ik})^{j^k}$ for all $j \in [n]$.

**Fig. 1.** The **DKG** protocol due to Gennaro et al.

to time period $t+1$, determine $C_{t+1}$ and compute the secret keys for all nodes in $C_{t+1} \backslash C_t$. Afterwards, delete $sk_{i,t}$ and all used re-randomization exponents $r_i''$.[5]

**FST.Encrypt**$(pk, t, M)$. Let $M \in \mathbb{G}_3$ be the message and $t_1 \dots t_\ell$ the bit representation of time period $t$. First, run **Sig.KeyGen**$\rightarrow$ $(vk, signk)$ and compute $H(vk) =: VK$. Then, pick a uniformly random $r \leftarrow \mathbb{Z}_q$ and compute $(C_1, C_2, C_3)$ as

$$\left( e(h, pk)^r \cdot M,\ g_2^r,\ (h_0 \prod_{v=1}^{\ell} h_v^{t_v} \cdot h_{\ell+1}^{VK})^r \right) \in \mathbb{G}_3 \times \mathbb{G}_2 \times \mathbb{G}_1$$

and **Sig.Sign**$(signk, (C_1, C_2, C_3), ) \rightarrow \sigma$. Output the ciphertext

$$C = (C_1, C_2, C_3), vk, \sigma).$$

**FST.Decrypt** $(t, (C_1, C_2, C_3), vk, \sigma)$. Let $\mathcal{W}$ be the set of indices of all participating parties. W.l.o.g. we assume that $\mathcal{W}$ contains at least $k+1$ distinct indices. The participating parties run the decryption protocol from Fig. 2.[6]

*Remark 1.* Note that the subset $\mathcal{V} \subseteq \mathcal{W}$ from the decryption protocol (Fig. 2) might be of size greater than $k+1$, while $k+1$ partial decryption shares are sufficient to decrypt the ciphertext. For this reason aggregating only $k+1$ decryption shares avoids computational overhead.

*Remark 2.* The secret keys in our fst-PKE have a binary tree structure in the sense of [6], except for the lowest level. In Theorem 4 from [6], it is shown that encryption schemes, which have a binary tree structure on all levels, imply forward security. Although it is possible to remove the lowest level and the strong one-time signature scheme in our construction, doing so would result in a scheme which is only forward secure against CPA instead of CCA.

**Proof of Correctness.** We have to prove that $(c', d')$ is a valid decryption key for ciphertext $C = (C_1, C_2, C_3, vk, \sigma)$ under $t, VK$ and the common public key

[5] Example: Let $T = 2^3$. Then $t_0 = 000$, $t_1 = 100$, $t_2 = 010$,... . Given a substring $xy$, we can compute $xy0$ and $xy1$. Hence, for time period $t_2$ the set $C_{t_2}$ consists of the node keys for 01 and 1. From 01 it can compute the secret key for $t_2 = 010$ and $t_3 = 011$. From 1 it can compute the secret key for all time periods greater $t_3$: 100, 101, 110, 111. The keys for time periods $t_0 = 000$ and $t_1 = 001$ cannot be computed from this set. If we update to time period $t_3$, we need to compute 011 and erase the node key for 01. Thus $C_{t_3}$ consists of the key for 011 and the node key 1. Then, also the key for 010 cannot be computed anymore.

[6] Note that decryption happens with respect to a time period. Since time periods are encoded in full bit length (even if they start with zero) they are low in the binary tree. Hence, they only have left $e_{i,\ell+1}$ as going down one level in depth erases one value $e_{i,x}, x \in [\ell]$.

$pk = g_2^x$. To that end we show first that set $\mathcal{V}$, which indicates the correct decryption shares, can be determined. That is, we show that we can check whether the decryption shares $(c_i', d_i')$, $i \in \mathcal{W}$ are correct. Let $(c_i', d_i')$ be an honestly generated decryption in Step 2 of the protocol. Then $d_i'$ is equal to

$$d_i e_{i,\ell+1}^{VK}(h_0 \prod_{v=1}^{\ell} h_v^{t_v} h_{\ell+1}^{VK})^{v_i} = h^{x_i}(h_0 \prod_{v=1}^{\ell} h_v^{t_v})^{r_i} e_{i,\ell+1}^{VK}(h_0 \prod_{v=1}^{\ell} h_v^{t_v} h_{\ell+1}^{VK})^{v_i}$$

$$= h^{x_i}(h_0 \prod_{v=1}^{\ell} h_v^{t_v} h_{\ell+1}^{VK})^{r_i+v_i}, \tag{5}$$

**Dec**$(t, (C_1, C_2, C_3), vk, \sigma, P_1, ..., P_n)$**:**

1. **Ciphertext-Verify.** At decryption request of $((C_1, C_2, C_3), vk, \sigma)$ at time $t = t_1 \dots t_\ell$, each party $P_i, i \in \mathcal{W}$ checks whether **Sig.Verify**$(vk, (C_1, C_2, C_3), \sigma) = 1$ and whether the ciphertext is valid for time period $t$ and for the hashed verification key $VK = H(vk)$. That is, it checks whether the following equation holds.

$$e((h_0 \prod_{v=1}^{s} h_v^{t_v} \cdot h_{\ell+1}^{VK}), C_2) = e(C_3, g_2).$$

   If one or both checks fail it aborts.
2. **Share-Decrypt.** Else each party picks a uniformly random $v_i \leftarrow \mathbb{Z}_q$ and uses its secret key $sk_{i,t} = (c_i, d_i, e_{i,\ell+1})$ to compute a decryption share $(c_i', d_i')$, where

$$d_i' := d_i \cdot e_{i,\ell+1}^{VK}(h_0 \prod_{v=1}^{\ell} h_v^{t_v} \cdot h_{\ell+1}^{VK})^{v_i} \quad \text{and} \quad c_i' := c_i \cdot g_2^{v_i}.$$

   Afterwards, each party $P_i, \in \mathcal{W}$ sends $(c_i', d_i')$ secretly to all other parties.
3. **Share-Verify.** All parties in $\mathcal{W}$ use the public keys $pk_j, j \in \mathcal{W}$ to check if the contributed decryption shares are valid. That is, if

$$e(d_j', g_2) = e(h, pk_j) \cdot e(h_0 \prod_{v=1}^{\ell} h_v^{t_v} \cdot h_{\ell+1}^{VK}, c_j').$$

4. **Combine.** Let $\mathcal{V} \subseteq \mathcal{W}$ indicate a set of parties sending valid decryption shares. If $\mathcal{V}$ contains at least $k+1$ distinct indices then the decryption key $(c', d')$ is computed as

$$c' = \prod_{i \in \mathcal{V}} c_i'^{L_i} \quad \text{and} \quad d' = \prod_{i \in \mathcal{V}} d_i'^{L_i},$$

   where $L_i = \prod_{j \in \mathcal{V}, j \neq i}(-i)/(j-i)$ are the Lagrange coefficients.
5. Finally, the plaintext is computed as

$$C_1 \cdot e(C_3, c')/e(d', C_2) = M. \tag{4}$$

**Fig. 2.** The **decryption** protocol of our fst PKE.

where we used the fact that $e_{i,\ell+1} = h_{i,\ell+1}^{r_i}$. Furthermore, $c_i' = g_2^{r_i+v_i}$. If a decryption share $(c_i', d_i')$ satisfies (5) and $c_i' = g_2^{r_i+v_i}$ then the validity check in Step 3 is correct, because

$$e(d_i', g_2) = e(h^{x_i}(h_0 \prod_{v=1}^{\ell} h_v^{t_v} h_{\ell+1}^{VK})^{r_i+v_i}, g_2)$$
$$= e(h^{x_i}, g_2)e((h_0 \prod_{v=1}^{\ell} h_v^{t_v} h_{\ell+1}^{VK})^{r_i+v_i}, g_2) = e(h, pk_i)e(h_0 \prod_{v=1}^{\ell} h_v^{t_v} h_{\ell+1}^{VK}, c_i').$$

For this reason, we can indeed check whether a decryption share $(c_i', d_i')$ for message $C$ under $t, VK$ and user public key $g_2^{x_i}$ is correct and thus include $i$ into set $\mathcal{V}$. It remains to show that all decryption shares $(c_i', d_i')$, $i \in \mathcal{V}$ interpolate to a valid decryption key under the common public key $g_2^x$. For this purpose, we set $R := \sum_{i\in\mathcal{V}} L_i(r_i + v_i)$. Then, $d'$ is equal to

$$\prod_{i\in\mathcal{V}} d_i'^{L_i} = h^{\sum_{i\in\mathcal{V}} L_i x_i}(h_0 \prod_{v=1}^{\ell} h_v^{t_v} h_{\ell+1}^{VK})^{\sum_{i\in\mathcal{V}} L_i(r_i+v_i)} = h^x(h_0 \prod_{v=1}^{\ell} h_v^{t_v} h_{\ell+1}^{VK})^R,$$

where $\sum_{i\in\mathcal{V}} L_i x_i = x$. Furthermore, $c' = \prod_{i\in\mathcal{V}} c_i'^{L_i} = g_2^{\sum_{i\in\mathcal{V}} L_i(r_i+v_i)} = g_2^R$. Overall, we have for a valid ciphertext

$$C_1 e(C_3, c')/e(d', C_2)$$
$$= Me(h, g_2^x)^r e((h_0 \prod_{v=1}^{\ell} h_v^{t_v} \cdot h_{\ell+1}^{VK})^r, g_2^R)/e(h^x(h_0 \prod_{v=1}^{\ell} h_v^{t_v} \cdot h_{\ell+1}^{VK})^R, g_2^r) = M.$$

**Proof of Security.** As preparation for the security proof we describe in Fig. 3 how the reduction simulates the **DKG** protocol. The simulation in Fig. 3 is also due to Gennaro et al. Additionally, Fig. 4 describes how the reduction simulates.

**FST.Decrypt.** According to Definition 4, the adversary is allowed to control up to $k$ parties during the key generation and decryption procedure. First, we assume a static adversary as in the proof in [10]. In the proof of Theorem 3, it is shown how to achieve security against adaptive adversaries. In Sect. 4, it is explained why this approach gives a more efficient scheme than the use of composite order groups as in [15]. W.l.o.g. we assume the corrupted parties to be $P_1, \ldots, P_k$. Let $\mathcal{B} := \{1, \ldots, k\}$ indicate the set of corrupted parties, controlled by the adversary $\mathcal{A}$, and let $\mathcal{G} := \{k+1, \ldots, n\}$ indicate the set of uncorrupted parties, run by the simulator.

Note that during the simulation of the decryption procedure the simulator already executed **DKGSim** (Fig. 3). Therefore, it is in possession of the secret shares $x_1, \ldots, x_k$ and the polynomials $a_i(z), b_i(z)$ for all $i \in [n]$.

**Protocol DKGSim**$(y = g_2^x, n, k)$**:**

1. The simulator performs Steps 1a−1f and 2 on behalf of the uncorrupted parties exactly as in the **DKG**$(n, k)$ protocol. Additionally, it reconstructs the polynomials $a_i(z), b_i(z)$ for $i \in \mathcal{B}$. Then:
   - The set *QUAL* is well-defined and $\mathcal{G} \subseteq QUAL$ and all polynomials are random for all $i \in \mathcal{G}$.
   - The adversary sees $a_i(z), b_i(z)$ for $i \in \mathcal{B}$, the shares $(s_{ij}, s'_{ij}) = (a_i(j), b_i(j))$ for $i \in QUAL$, $j \in \mathcal{B}$ and $C_{is}$ for $i \in QUAL$, $s = 0, \ldots, k$.
   - The simulator knows all polynomials $a_i(z), b_i(z)$ for $i \in QUAL$ as well as all shares $s_{ij}, s'_{ij}$, all coefficients $a_{is}, b_{is}$ and the public values $C_{is}$.
2. The simulator performs as folllows:
   - Computes $A_{is} = g_2^{a_{is}} \in \mathbb{G}_2$ for $i \in QUAL \setminus \{n\}$, $s = 0, \ldots, k$.
   - Sets $A^*_{n0} = y \cdot \prod_{i \in QUAL \setminus \{n\}} (A_{i0}^{-1})$.
   - Sets $s^*_{nj} = s_{nj} = a_n(j)$ for $j = 1, \ldots, k$.
   - Computes $A^*_{ns} = (A^*_{n0})^{\lambda_{s0}} \cdot \prod_{i=1}^{k} (g_2^{s^*_{ni}})^{\lambda_{si}} \in \mathbb{G}_2$ for $s = 1, \ldots, k$, where the $\lambda_{is}$s are the Lagrange interpolation coefficients.

   (a) The simulator broadcasts $A_{is}$ for $i \in \mathcal{G} \setminus \{n\}$ and $A^*_{ns}$ for $s = 0, \ldots, k$.
   (b) It performs for all uncorrupted parties the verification of (2) on the values $A_{ij}$ for $i \in \mathcal{B}$. In case of a fail it broadcasts a complaint $(s_{ij}, s'_{ij})$. Since the adversary controls at most $k$ parties and the simulator behaves honestly, only secret shares of corrupted parties can be reconstructed.
   (c) Afterwards it performs the Steps 4c and 4d of the **DKG**$(n, k)$ protocol.

**Fig. 3.** The simulation of the **DKG** protocol due to Gennaro et al.

In the **DKG** protocol (Fig. 1) the secret shares for all parties $P_j, j \in [n]$ are defined as $x_j := \sum_{i \in QUAL} s_{ij} \mod q$. In **DKGSim** however, the shares are defines as $x_j := \sum_{i \in QUAL \setminus \{n\}} s_{ij} + s^*_{nj} \mod q$, where the values $s^*_{nj}$ for $j = k+1, \ldots, n$, i.e. for $j \in \mathcal{G}$, are not explicitly known. Moreover, in **DKGSim** the public key of user $P_j, j \in [n]$ is $g_2^{x_j} = \prod_{i \in QUAL \setminus \{n\}} g_2^{s_{ij}} g_2^{s^*_{nj}} = \prod_{i \in QUAL \setminus \{n\}} \prod_{s=0}^{k} (A_{is})^{j^s} \prod_{s=0}^{k} (A^*_{ns})^{j^s}$, where the values $A^*_{ns}$ include the common public key $y = g_2^x$. Hence, in order to compute the secret share $h^{x_j}$ for $j \in \mathcal{G}$ either the corresponding value $h^x$ or $s^*_{nj}$ is required. Although these values are not known to the simulator it is still able to simulate the role of the uncompromised parties during the decryption of a valid ciphertext $(t, C_1, C_2, C_3, vk, \sigma)$. In order to do so it requires a valid secret key $(c, d)$ for time $t$ together with the hashed verification key $VK = H(vk)$, i.e. for the string $t, VK$. If the simulator is an adversary breaking the CPA security of the HIBE scheme from [2] this key can be requested in its own security experiment. Let $t = t_1 \ldots t_\ell$ and $(c, d) = \left(g_2^r, h^x (h_0 \prod_{v=1}^{\ell} h_v^{t_v} \cdot h_{\ell+1}^{VK})^r\right)$. Define consistently with **DKGSim**:

- $H_{is} := h^{a_{is}}$ for all $i \in QUAL \setminus \{n\}, s = 0, \ldots, k$
- $H^*_{n0} := \prod_{i \in QUAL \setminus \{n\}} (H_{i0}^{-1}) h^x$

- $s^*_{nj} := s_{nj} = a_n(j)$ for $j = 1, \ldots, k$
- $H^*_{ns} := (H^*_{n0})^{\lambda_{s0}} \prod_{i=1}^{k} (h^{s^*_{ni}})^{\lambda_{si}}$ for $s = 1, \ldots, k$
- $\hat{H}_{n0} := \prod_{i \in QUAL \setminus \{n\}} (H_{i0}^{-1} d)$.

Thus, $pk_j = h^{x_j}, j \in \mathcal{G}$ are defined as $\prod_{i \in QUAL \setminus \{n\}} \prod_{s=0}^{k} (H_{is})^{j^s} \prod_{s=0}^{k} (H^*_{ns})^{j^s}$. To obtain a valid decryption share for $P_j, j \in \mathcal{G}$ compute an intermediate $d''_j$ as

$$\prod_{i \in QUAL \setminus \{n\}} \prod_{s=0}^{k} (H_{is})^{j^s} (\hat{H}_{n0}) \prod_{s=1}^{k} \left( (\hat{H}_{n0})^{\lambda_{s0}} \prod_{i=1}^{k} (h^{s^*_{ni}})^{\lambda_{si}} \right)^{j^s}$$

$$= \prod_{i \in QUAL \setminus \{n\}} \prod_{s=0}^{k} (H_{is})^{j^s} \left( \prod_{i \in QUAL \setminus \{n\}} (H_{i0}^{-1}) h^x (h_0 \prod_{v=1}^{\ell} h_v^{t_v} h_{\ell+1}^{VK})^r \right)$$

$$\prod_{s=1}^{k} \left( ( \prod_{i \in QUAL \setminus \{n\}} (H_{i0}^{-1}) h^x (h_0 \prod_{v=1}^{\ell} h_v^{t_v} h_{\ell+1}^{VK})^r )^{\lambda_{s0}} \prod_{i=1}^{k} (h^{s^*_{ni}})^{\lambda_{si}} \right)^{j^s}$$

$$= \prod_{i \in QUAL \setminus \{n\}} \prod_{s=0}^{k} (H_{is})^{j^s} \left( \prod_{i \in QUAL \setminus \{n\}} (H_{i0}^{-1}) h^x \right) (h_0 \prod_{v=1}^{\ell} h_v^{t_v} h_{\ell+1}^{VK})^r$$

$$\prod_{s=1}^{k} \left( ( \prod_{i \in QUAL \setminus \{n\}} (H_{i0}^{-1}) h^x )^{\lambda_{s0}} \prod_{i=1}^{k} (h^{s^*_{ni}})^{\lambda_{si}} \right)^{j^s} \prod_{s=1}^{k} \left( (h_0 \prod_{v=1}^{\ell} h_v^{t_v} h_{\ell+1}^{VK})^{r \lambda_{s0}} \right)^{j^s}$$

$$= \prod_{i \in QUAL \setminus \{n\}} \prod_{s=0}^{k} (H_{is})^{j^s} ( \prod_{s=0}^{k} (H^*_{ns})^{j^s} ) (h_0 \prod_{v=1}^{\ell} h_v^{t_v} h_{\ell+1}^{VK})^{r \sum_{s=1}^{k} \lambda_{s0} j^s + r}$$

$$= h^{x_j} (h_0 \prod_{v=1}^{\ell} h_v^{t_v} h_{\ell+1}^{VK})^{r \sum_{s=1}^{k} \lambda_{s0} j^s + r}.$$

To re-randomize, pick a uniformly random $w_j \leftarrow \mathbb{Z}_p$ and compute $d'_j$ and $c'_j$ as

$$d''_j (h_0 \prod_{v=1}^{\ell} h_v^{t_v} h_{\ell+1}^{VK})^{w_j} \text{ and } c^{\sum_{s=1}^{k} \lambda_{s0} j^s + 1} g_2^{w_j}. \tag{6}$$

**Lemma 1.** *The protocols **DKG** and **DKGSim** as well as **Dec** and **DecSim** are indistinguishable.*

*Proof.* For **DKG** and **DKGSim** the proof can be found in Theorem 2 of [10]. In the same fashion it can be easily verified that the adversary has the same view **Dec** and **DecSim**. □

**Theorem 1.** *The scheme $(T, n, k)$-$\Pi_{fst}$ from Sect. 3 is $(n, k_1, k_2)$-robust if $k_1 + k_2 \leq k$ and $n \geq 2k + 1$. In particular, the scheme is $(n, 0, k)$-robust, i.e. robust against malicious adversaries.*

**DecSim**$(t, (C_1, C_2, C_3), vk, \sigma), (c, d))$

1. The decryption is requested. Let $t = t_1 \dots t_\ell$. The simulator performs the validity checks from Step 1 of protocol **Dec**. If one of these checks fail it aborts.
2. Else, the simulator uses the polynomials $a_i(z), b_i(z)$ for all $i \in [n]$ and the decryption key $(c, d)$ for $t, VK$ to compute valid decryption shares on behalf of all uncorrupted parties: for $j = k+1, \dots, n$ it computes $d''_j$ as

$$\prod_{i \in QUAL \setminus \{n\}} \prod_{s=0}^{k} (H_{is})^{j^s} \cdot \left(\hat{H}_{n0}\right) \cdot \prod_{s=1}^{k} \left((\hat{H}_{n0})^{\lambda_{s0}} \cdot \prod_{i=1}^{k} (h^{s^*_{ni}})^{\lambda_{si}}\right)^{j^s}.$$

   Then, it picks $w_j \leftarrow \mathbb{Z}_q$ uniformly at random and computes $d'_j$ as

$$d''_j \cdot (h_0 \prod_{v=1}^{\ell} h_v^{t_v} \cdot h_{\ell+1}^{VK})^{w_j}.$$

   Afterwards, it computes $c'_j$ as:

$$c^{\sum_{s=1}^{k} \lambda_{s0} \cdot j^s + 1} \cdot g_2^{w_j}.$$

   Then, the simulator sends $(c'_j, d'_j)$ for all $j = k+1, \dots, n$ to the corrupted parties $P_1, \dots, P_k$. The simulator might receive decryption shares on behalf of the corrupted parties.
3. The simulator does nothing.
4. The adversary can use any set $\mathcal{V}$ of at least $k+1$ partial decryption shares to construct the final decryption key:

$$(c', d') = (\prod_{i \in \mathcal{V}} c_i^{L_i}, \prod_{i \in \mathcal{V}} d_i^{L_i}),$$

   where $L_i = \prod_{j \in \mathcal{V}, j \neq i} (-i)/(j-i)$ are Lagrange coefficients. The simulator does nothing.
5. The adversary can use the decryption key $(c, d)$ to decrypt the ciphertext. The simulator does nothing.

**Fig. 4.** The simulation of the **decryption** protocol.

*Proof.* We argue for the strongest case, i.e. $(n, 0, k)$. To show that $\Pi_{fst}$ is $(n, 0, k)$-robust we analyze all protocols where the adversary on behalf of the uncompromised parties may interact with the honest ones, i.e. **FST.KeyGen** and **FST.Decrypt**. More precisely, we show that the adversary is incapable to prevent the honest parties from executing these protocols successfully. The **FST.KeyGen** protocol is instantiated with the **DKG** protocol from [10], which was shown to be robust against malicious adversaries. The reason for this is that a party which deviates from the protocol specification is either disqualified or its secret share is reconstructed by the honest parties. In the case of the

**FST.Decrypt** protocol the adversary has two options to attempt cheating. One option is to try manipulating the ciphertext. This however, is prevented in Step 1 of the decryption protocol by checking the ciphertext for validity. The second option is to try manipulating or denying decryption shares. This is prevented in Step 3 by using the user public keys $g^{x_i}, i \in [n]$ to check if the decryption shares are valid. Hence only valid decryption shares are aggregated and the message is decrypted correctly. Moreover, the adversary is allowed to control or halt at most $k$ parties. Thus, a valid decryption share can still be computed as long as $n \geq 2k + 1$. □

**Theorem 2.** *Let $n \geq 2k + 1$ and let $\mathcal{A}$ be a static adversary that $(t_{\mathcal{A}}, \varepsilon_{\mathcal{A}})$-breaks the CCA forward security of $(T, n, k)$-$\Pi_{fst}$ from Sect. 3. Given $\mathcal{A}$, we can build an adversary $\mathcal{A}'$ that $(t_{\mathcal{A}'}, \varepsilon_{\mathcal{A}'})$-breaks the CPA security of HIBE $\Pi_{HIBE}$ from [2], an adversary $\mathcal{A}''$ that $(t_{\mathcal{A}''}, \varepsilon_{\mathcal{A}''})$-breaks the sEUF-1CMA security of a signature scheme $\Sigma$, and an adversary $\mathcal{A}'''$ that $(t_{\mathcal{A}'''}, \varepsilon_{\mathcal{A}'''})$-breaks the collision resistance of hash function $H$, such that*

$$t_{\mathcal{A}'''} \approx t_{\mathcal{A}''} \approx t_{\mathcal{A}'} \approx t_{\mathcal{A}} \text{ and } \varepsilon_{\mathcal{A}'''} + \varepsilon_{\mathcal{A}''} + \varepsilon_{\mathcal{A}'} \geqslant \varepsilon_{\mathcal{A}}.$$

□

*Proof.* Conceptually, we follow the proofs from Sections. 4 and 6 in [4], which were also reproduced in Section 4.1 in [3]. In [4], a CPA-secure HIBE with $\ell + 1$ levels and identities of length $n + 1$ bits is turned into a CCA-secure HIBE with $\ell$ levels and identities of length $n$ bits. The reason for the shorter identities in the CCA-secure scheme is that this framework uses one bit of the identity as a padding. This padding guarantees that decryption queries do not correspond to prefixes of the challenge identity. In our scheme however, the first $\ell$ levels are single bits and the deepest level has elements in $\mathbb{Z}_q$, which makes it impossible to spend one bit of each identity for the padding. However, in our scheme the adversary is only allowed to make decryption queries with respect to time periods (plus a value in $\mathbb{Z}_q$). Since time periods are always encoded with full length they cannot correspond to prefixes of each other. Thus a padding is not necessary.

We start with describing an adversary $\mathcal{A}'$ playing the CPA security game for HIBE $\Pi_{HIBE}$ and simulating the CCA forward security game for a static adversary $\mathcal{A}$.

At the beginning, $\mathcal{A}$ sends its choice of the $k$ parties it wants to corrupt to $\mathcal{A}'$. Let $\mathcal{B}$ denote the set of the indices of these parties and $\mathcal{G} := \{1, \ldots, n\} \setminus \mathcal{B}$. Adversary $\mathcal{A}'$ runs **Sig.KeyGen** to obtain $(vk^*, signk^*)$. Then it computes $H(vk^*) := VK^*$. Moreover, it receives a master public key $mpk := g_2^x \in \mathbb{G}_2$ from its own security experiment. In order to simulate the **FST.KeyGen** procedure adversary $\mathcal{A}'$(on behalf of the uncompromised parties) runs the **DKGSim** protocol on input $(g_2^x, n, k)$. Both adversaries receive all information to compute the secret key shares of all compromised parties and the user public keys $pk_i$ for all $i \in [n]$ as well as the common public key $pk = g_2^x$. Afterwards, $\mathcal{A}$ has access to the following procedures, which are simulated by $\mathcal{A}'$ as follows.

**Break-In***(t',j).* On input a time period $t' = t_1 \dots t_\ell$ adversary $\mathcal{A}'$ queries **KeyQuery** on all nodes from the set $C_{t'}$, which was defined in the **KeyUpdate** procedure in 3. According to the definition of $C_{t'}$ these are all the nodes which allow the computation of the secret keys for all time periods $t \geqslant t'$ but for no time period $t < t'$. As a response it obtains tuples of the form

$$(c, d, e_{s+1}, \dots, e_{\ell+1}) = \left( g_2^r, h^x (h_0 \prod_{v=1}^{s} h_v^{w_v})^r, h_{s+1}^r, \dots, h_{\ell+1}^r \right),$$

which correspond to internal nodes $\omega = \omega_1 \dots \omega_s$, where $s \leq \ell$. In order to compute the secret keys $sk_{t',j}$ for all $j \in \mathcal{G}$ it proceeds as follows. It defines equivalently to **DecSim**:

- $H_{is} := h^{a_{is}}$ for all $i \in QUAL \setminus \{n\}, s = 0, \dots, k$
- $H^*_{n0} := \prod_{i \in QUAL \setminus \{n\}} (H_{i0}^{-1}) \cdot h^x$
- $s^*_{nj} := s_{nj} = a_n(j)$ for $j = 1, \dots, k$
- $H^*_{ns} := (H^*_{n0})^{\lambda_{s0}} \cdot \prod_{i=1}^{k} (h^{s^*_{ni}})^{\lambda_{si}}$ for $s = 1, \dots, k$
- $\bar{H}_{n0} := \prod_{i \in QUAL \setminus \{n\}} (H_{i0}^{-1} \cdot d)$ for each tuple $(c, d, \dots)$ separately.

It computes for all tuples a corresponding value $d_j$ as:

$$\prod_{i \in QUAL \setminus \{n\}} \prod_{s=0}^{k} (H_{is})^{j^s} \cdot \left(\bar{H}_{n0}\right) \cdot \prod_{s=1}^{k} \left( (\bar{H}_{n0})^{\lambda_{s0}} \cdot \prod_{i=1}^{k} (h^{s^*_{ni}})^{\lambda_{si}} \right)^{j^s}$$

and for all values $\tilde{x}$ from $\{c, e_{i+1}, \dots, e_{\ell+1}\}$ it computes $\tilde{x}_j$ as $\tilde{x}^{\sum_{s=1}^{k} \lambda_{s0} \cdot j^s + 1}$.

In order to guarantee a perfect simulation $\mathcal{A}'$ re-randomizes the secret keys of all parties in the same fashion as the decryption shares in (6). Finally, it outputs $sk_{t',j}$ for all $j \in \mathcal{G}$ as the stack of tuples of the form $(c_j, d_j, e_{j,i+1}, \dots, e_{j,\ell+1})$.

Analogously to the decryption in **DecSim** it holds that

$$d_j = h^{x_j} \cdot (h_0 \prod_{v=1}^{s} h_v^{w_v})^{r' \cdot \sum_{s=1}^{k} \lambda_{k0} \cdot j^s + r'}.$$

Overall, these stacks form valid secret keys $sk_{t',j}$ for all $j \in \mathcal{G}$ and their simulation is perfect. If **Challenge**$(t^*, M_0, M_1)$ was already queried then all break-in queries with $t' \leqslant t^*$ are invalid.

**Challenge**$(t^*, M_0, M_1)$. Adversary $\mathcal{A}$ submits two messages $M_0, M_1$ and challenge time period $t^*$. Adversary $\mathcal{A}'$ forwards $(t^*.VK^*, M_0, M_1)$ to **Challenge** in its own security game and receives a ciphertext $(C_1, C_2, C_3)$ which equals

$$\left( e(h, pk)^r \cdot M_b,\ g_2^r,\ (h_0 \prod_{v=1}^{\ell} h_v^{t_v} \cdot h_{\ell+1}^{VK^*})^r \right) \in \mathbb{G}_3 \times \mathbb{G}_2 \times \mathbb{G}_1,$$

where $b$ is a uniformly random bit. Afterwards, it computes a signature $\sigma^* \leftarrow$ **Sig.Sign**$(signk^*, (C_1, C_2, C_3))$ and outputs the challenge ciphertext $C^* = (C_1,$

$C_2, C_3, vk^*, \sigma^*)$. If **Break-In** on input $t' \leqslant t^*$ was previously queried then the challenge query is invalid.
**Dec**$(t, C_1, C_2, C_3, vk, \sigma)$. Whenever $\mathcal{A}$ asks for a decryption then $\mathcal{A}'$ proceeds as follows. First, it checks if $vk = vk^*$ and **Sig.Verify**$(vk, (C_1, C_2, C_3), \sigma) = 1$ or if $vk \neq vk^*$ and $H(vk) = VK^*$. If one of these conditions is true then $\mathcal{A}'$ aborts and outputs a uniformly random bit to **Guess** in its own security game. Else it queries **KeyQuery**$(t, VK)$ to obtain the decryption key $sk_{t,VK} = (c, d)$. Then, it simulates the decryption procedure by running **DecSim**$(t, (C_1, C_2, C_3), vk, \sigma, (c, d))$. Since $t, VK$ is unequal to and no prefix of $t^*.VK^*$ the query to **KeyQuery** is valid. **Dec** cannot be queried on input $(t^*, C^*)$.
**Guess***(b')*. Adversary $\mathcal{A}$ outputs its guess $b' \in \{0, 1\}$, which $\mathcal{A}'$ forwards to **Guess** in its own experiment.

We denote **Forge** the event that $\mathcal{A}'$ aborts during a decryption query because of the first condition and **Coll** that it aborts because of the second condition. Together with Lemma 1 it can be seen that adversary $\mathcal{A}'$ provides a perfect simulation to $\mathcal{A}$ as long as any of these two events do *not* happen. Thus,

$$|\varepsilon_{\mathcal{A}} - \varepsilon_{\mathcal{A}'}| \leqslant \Pr[\mathbf{Forge} \cup \mathbf{Coll}] = \Pr[\mathbf{Forge}] + \Pr[\mathbf{Coll}]. \tag{7}$$

In order to determine $\Pr[\mathbf{Forge}]$ note that if **Forge** occurs then $\mathcal{A}$ has submitted a valid ciphertext $(C_1, C_2, C_3, vk^*, \sigma^*)$, which means that $\sigma^*$ is a valid signature for message $(C_1, C_2, C_3)$ under verification key $vk^*$. We show how to build an adversary $\mathcal{A}''$ that breaks the sEUF-1CMA security of $\Sigma$ using $\mathcal{A}$.

Adversary $\mathcal{A}''$ plays the sEUF-1CMA security game with respect to $\Sigma$. At the beginning, it receives a verification key $vk^*$ from its challenger. After $\mathcal{A}$ has submitted its choice of corrupted parties adversary $\mathcal{A}''$ picks a uniformly random $x \leftarrow \mathbb{Z}_q$ and executes **DKGSim**$(g_2^x, n, k)$ on behalf of the uncorrupted parties. Since $\mathcal{A}''$ is in possession of $x$ it is able to simulate all secret keys queried to **Break-In**. If $\mathcal{A}$ makes a valid query **Dec**$(t, C_1, C_2, C_3, vk^*, \sigma^*)$ then $\mathcal{A}''$ outputs $(C_1, C_2, C_3, \sigma^*)$ as a forgery to its own security experiment. If $\mathcal{A}$ makes a query **Challenge**$(t^*, M_0, M_1)$ then $\mathcal{A}''$ picks a bit $b$ uniformly at random and computes **FST.Encrypt**$(pk, t^*, M_b) \rightarrow (C_1, C_2, C_3)$. Afterwards, it queries the signing oracle in its own security experiment on input $(C_1, C_2, C_3)$. It receives a signature $\sigma$ and returns $(t^*, C_1, C_2, C_3, vk^*, \sigma)$ to $\mathcal{A}$. If $\mathcal{A}$ happens to query **Dec**$(C_1, C_2, C_3, vk^*, \sigma^*)$ then, $\mathcal{A}''$ submits $((C_1, C_2, C_3), \sigma^*)$ as its forgery. Note that if the challenge oracle was already queried we still have $((C_1, C_2, C_3), \sigma) \neq ((C_1, C_2, C_3), \sigma^*)$. It follows that

$$\Pr[\mathbf{Forge}] = \varepsilon_{\mathcal{A}''}. \tag{8}$$

It remains to determine $\Pr[\mathbf{Coll}]$ for $H$ by building an adversary $\mathcal{A}'''$ that breaks the collision resistance of $H$. It is easy to see that adversary $\mathcal{A}'''$ can simulate the CCA forward security game for $\mathcal{A}$ perfectly by running **FST.KeyGen** and **Sig.KeyGen**. Whenever a collision occurs it forwards the corresponding inputs to $H$ to its own challenger. It follows that

$$\Pr[\mathbf{Coll}] = \varepsilon_{\mathcal{A}'''}. \tag{9}$$

Putting (8) and (9) in (7) gives us $\varepsilon_{\mathcal{A}} \leqslant \varepsilon_{\mathcal{A}'} + \varepsilon_{\mathcal{A}''} + \varepsilon_{\mathcal{A}'''}$.

It is easy to see that all algorithms run in approximately the same time. This completes the proof. □

**Theorem 3.** *Let $n \geq 2k+1$ and let $\mathcal{A}$ be an adaptive adversary that $(t_{\mathcal{A}}, \varepsilon_{\mathcal{A}})$-breaks the CCA forward security of $(T, n, k)$-$\Pi_{fst}$ from Sect. 3. Given $\mathcal{A}$ we can build an adversary $\mathcal{A}'$ that $(t_{\mathcal{A}'}, \varepsilon_{\mathcal{A}'})$-breaks the CPA security of HIBE $\Pi_{HIBE}$ from [2], an adversary $\mathcal{A}''$ that $(t_{\mathcal{A}''}, \varepsilon_{\mathcal{A}''})$-breaks the sEUF-1CMA security of a signature scheme $\Sigma$, and an adversary $\mathcal{A}'''$ that $(t_{\mathcal{A}'''}, \varepsilon_{\mathcal{A}'''})$-breaks the collision resistance of hash function $H$, such that*

$$t_{\mathcal{A}'''} \approx t_{\mathcal{A}''} \approx t_{\mathcal{A}'} \approx t_{\mathcal{A}} \text{ and } \varepsilon_{\mathcal{A}'''} + \varepsilon_{\mathcal{A}''} + \binom{n}{k} \cdot \varepsilon_{\mathcal{A}'} \geqslant \varepsilon_{\mathcal{A}}.$$

*Proof.* Adversary $\mathcal{A}'$ proceeds as in the proof of Theorem 2. The only difference is that it guesses in advance of step 1 of **DKGSim** which parties the adversary is going to corrupt. Whenever the adversary corrupts a party $P_j$, $j \in \mathcal{B}$ then it takes over the role of this party and receives all values computed and stored on behalf of this party by $\mathcal{A}'$. If at the end $\mathcal{A}$ outputs a bit but has not corrupted at least $k$ parties then $\mathcal{A}'$ adds some artificial corruptions to the set $\mathcal{B}$ uniformly at random such that it has exactly $k$ corrupted parties. Adversary $\mathcal{A}'$ aborts the simulation and outputs uniformly random bit if a guess was wrong (either of $\mathcal{A}$ or $\mathcal{A}'$). The simulation is successful with probability $1/\binom{n}{k}$.

Adversary $\mathcal{A}''$ also proceeds as in the proof of Theorem 2. Since it is in possession of the common secret $x$ it can compute the secret values $h^{x_i}$ for all $i \in [n]$ directly. Hence, it has no additional loss in its success probability. Guessing the corrupted parties has also no effect on breaking collision resistance and thus its probability remains unchanged as well.

## 4 Discussions

*Tolerating Mobile Adversaries.* A mobile adversary is able to switch between the parties it corrupts. It holds in general that it is not possible to tolerate such adversaries while having a non-interactive key update procedure. The reason is that a mobile adversary could gain all secret key shares successively without ever exceeding the threshold in any time period. Then, by updating all the shares to the latest time period it would be able to reconstruct the secret key. However, in our fst-PKE prevention against mobile adversaries is possible by adding a proactive security mechanism [11,12,17]. Proactive security allows to refresh the secret key shares in a way such that all shares which were *not* refreshed cannot be used to reconstruct the secret key anymore (except for the case that the amount of not refreshed shares is bigger than the threshold). Although the proactive security mechanism is *interactive*, the secret share holders can decide how often or when they are willing to execute it. For instance, this could happen with a different level of granularity than the non-interactive key update mechanism or only when necessary. In order to proactivize the key material in our scheme it

does not even require an additional protocol. Indeed, it suffices that the users execute the **DKG** protocol where each party $P_i$ sets the constant term of polynomial $a_i$ to 0. Then, the final share held by all parties is multiplied to all terms $d'_i$ in their secret key shares.

*From Static to Adaptive Adversaries.* To protect against adaptive adversaries we used complexity leveraging. This approach results in an additional security loss of $\binom{n}{k}$, where $n$ is the number of parties and $k$ the threshold. Although, this loss seems to be quite big, in practice $n$ is relatively small. For instance for a threshold scheme with $10, 20$ or $30$ parties the maximum loss is $2^8, 2^{18}$ and $2^{28}$, respectively. Libert and Yung [15] also achieve security against adaptive adversaries but circumvent complexity leveraging by using bilinear pairings of composite order. This approach is known as the dual system approach and prior to their work it was only used to achieve full security for (Hierarchical-)IBE and attribute-based encryption schemes [13,14,19]. Although the dual system approach is a very powerful tool to obtain full security or security against adaptive adversaries it lacks efficiency when implemented. The reason for this is that groups of composite order require a much bigger modulus to guarantee the same level of security than elliptic curves on groups of prime order.[7] It can be seen that the security loss in our scheme can be compensated by a slightly bigger modulus. This modulus remains much smaller compared to one of composite order and thus results in a much more efficient scheme.

Finally, it should be mentioned that there exist several techniques to transfer the dual system approach to prime order groups [9,14,19]. However, they result in larger ciphertexts and seem also to be less efficient in terms of communication rounds for decryption. Moreover, it is not clear whether they can be instantiated without a trusted dealer. We leave it as an open problem to use these techniques to achieve the same efficiency and advantages as our fst-PKE, i.e. a *non-interactive* key update and decryption procedure, *no trusted dealer*, and the possibility to implement the scheme on *standardized elliptic curves*.

## References

1. Abdalla, M., Miner, S., Namprempre, C.: Forward-Secure threshold signature schemes. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 441–456. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45353-9_32
2. Boneh, D., Boyen, X., Goh, E.-J.: Hierarchical identity based encryption with constant size ciphertext. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_26
3. Boneh, D., Boyen, X., Halevi, S.: Chosen ciphertext secure public key threshold encryption without random oracles. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 226–243. Springer, Heidelberg (2006). https://doi.org/10.1007/11605805_15

[7] For comparison of concrete sizes see the common recommendations: https://www.keylength.com/.

4. Boneh, D., Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. SIAM J. Comput. **36**(5), 1301–1328 (2007)
5. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_30
6. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 255–271. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_16
7. Chatterjee, S., Hankerson, D., Menezes, A.: On the efficiency and security of pairing-based protocols in the type 1 and type 4 settings. In: Hasan, M.A., Helleseth, T. (eds.) WAIFI 2010. LNCS, vol. 6087, pp. 114–134. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13797-6_9
8. Chow, S.S.M., Go, H.W., Hui, L.C.K., Yiu, S.-M.: Multiplicative forward-secure threshold signature scheme. Int. J. Netw. Secur. **7**, 397–403 (2008)
9. Freeman, D.M.: Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 44–61. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_3
10. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. J. Cryptology **20**(1), 51–83 (2006). https://doi.org/10.1007/s00145-006-0347-3
11. Herzberg, A., Jakobsson, M., Jarecki, S., Krawczyk, H., Yung, M.: Proactive public key and signature systems. In: Proceedings of the ACM Conference on Computer and Communications Security, January 1997
12. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or: how to cope with perpetual leakage. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 339–352. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-44750-4_27
13. Lewko, A., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully secure functional encryption: attribute-based encryption and (Hierarchical) inner product encryption. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 62–91. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_4
14. Lewko, A., Waters, B.: New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 455–479. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11799-2_27
15. Libert, B., Yung, M.: Adaptively secure non-interactive threshold cryptosystems. Theoret. Comput. Sci. **478**, 76–100 (2013)
16. Liu, L.-S., Chu, C.-K., Tzeng, W.-G.: A threshold GQ signature scheme. In: Zhou, J., Yung, M., Han, Y. (eds.) ACNS 2003. LNCS, vol. 2846, pp. 137–150. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45203-4_11
17. Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks (extended abstract). In: Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing, PODC 1991, pp. 51–59. ACM, New York (1991)
18. Tzeng, W.-G., Tzeng, Z.-J.: Robust forward-secure signature schemes with proactive security. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 264–276. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44586-2_19
19. Waters, B.: Dual system encryption: realizing fully secure IBE and HIBE under simple assumptions. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 619–636. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_36

20. Yu, J., Kong, F.: Forward secure threshold signature scheme from bilinear pairings. In: Wang, Y., Cheung, Y., Liu, H. (eds.) CIS 2006. LNCS (LNAI), vol. 4456, pp. 587–597. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74377-4_61
21. Zhang, X., Xu, C., Zhang, W.: Efficient chosen ciphertext secure threshold public-key encryption with forward security. In: Proceedings of the 2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies, EIDWT 2013, USA, pp. 407–413. IEEE Computer Society (2013)