Distributed Software Leasing

Nikhil Pappu[†]

[†]Portland State University
nikpappu@pdx.edu

December 13, 2024

Abstract

Contents

1 Introduction

In Secure Software Leasing (SSL) with finite-term security, a function can be leased to a server such that the server is able to evaluate the function an unbounded polynomially many times. Additionally, the server can later be asked to revoke its lease in a way that passes certain verification. It is guaranteed that upon successful verification, the server no longer has the ability to evaluate the function. Clearly, if a function is learnable (it can be replicated given only black-box access), then it cannot be securely leased. Consequently, one might hope to target the class of unlearnable functions. However, it was shown in the work of Ananth and LaPlaca [AL21] that there exist unlearnable functions for which this notion is impossible to achieve. In this work, we explore the setting where a client can lease out a function to two servers S_0 and S_1 in a distributed sense, such that they can compute additive shares of evaluations of the function. We will consider the following security notion:

Finite-Term Security: This security notion guarantees that the following events cannot both happen:

1. Either server provides a certificate of revocation that passes verification.
2. S_0, S_1 produce quantum states ρ_0, ρ_1 which are given to a QPT adversary $\mathcal{A}$. Given a random input x^*, $\mathcal{A}$ produces the correct output of the software on x^*.

Other than finite-term security, the stronger notions of infinite-term security and copy-protection have also been studied in the literature. These notions do not involve deletion certificates and only require the adversary to produce two working copies of the software. While infinite-term security requires the copies to work with the honest evaluation algorithm, copy protection is even stronger and permits arbitrary evaluations on the copies. Later on, we will also explore extensions of these notions to the distributed setting.

1.1 Motivation

The motivation for the different aspects of the notion are as follows:

- **Additive Reconstruction:** The requirement of additive secret-sharing is highly desirable as opposed to arbitrary (potentially even quantum) reconstruction. Recall that in the case of SSL, a single party obtains evaluations in the clear. In applications where a server obtains different software from several clients and needs to compute on the individual outputs, it can do so easily. This is non-trivial in the distributed setting. Additive secret-shares offer a middle ground by allowing linear computations on the outputs for free. Moreover, they are in the correct format for MPC protocols which can allow for more general computations. Furthermore, additive reconstruction enables efficient reconstruction, both in terms of computation and communication.
- **Finite-Term Security:** It can be reasonable to assume that the servers do not collude in the short term, but over time they may collude or an adversary may get access to their data. This security notion guarantees that such a data breach from both servers does not grant the ability to evaluate the software.
- **Secrecy:** Apart from finite term security, we also require that if the servers do not collude, they do not learn anything about the leased program. This requirement is similar to that of Function Secret Sharing (FSS). This provides a two-tier security guarantee. For instance, the servers can be used to delegate a decryption functionality without them being able to decrypt themselves in real time. Although they may decrypt past ciphertexts after collusion, that data may not be very relevant later on.

Consider now the following example use case for this notion:

Use Case. We will consider a delegation scenario where a client needs to delegate a function f. Of course, it can use an FHE ciphertext and send it to a single server but this complicates the decryption procedure. Also, if the client's secret-key is leaked at any point in the future, the function can be completely learnt. More importantly though, multiple FHE ciphertexts cannot be evaluated on and compressed easily, which can be especially important when multiple clients are involved. One could also consider Multi-Key FHE which would not only be computationally expensive, but would need distributed decryption. In such a scenario, Function Secret Sharing (FSS) can prove to be a good alternative due to the additive reconstruction property, albeit at the cost of utilizing two servers. However, there is now the possibility that in the long term, the servers may be breached, providing the adversary with the ability to evaluate the software. Hence, DSL can also be viewed as an extension of FSS to the quantum setting, where it is upgraded with an SSL like guarantee.

2 To Do

1. [$*$] DOTP Redn to LOTA
2. [ ] Punc DSL Formal Construction + Theorem
3. [ ] Punc DSL Redn to dIO-CD

3 Definitions of Distributed Software Leasing

3.1 Syntax of DSL

A scheme satisfying the DSL syntax for a PPT computable circuit class $\mathcal{C}$ is a tuple of 4 algorithms $\mathsf{DSL} = \mathsf{DSL}.(\mathcal{Share}, \mathcal{Eval}, \mathcal{Del}, \mathsf{Vrfy})$ with the following properties:

Syntax:

$\mathcal{Share}(P) \to (\mathcal{P}_0, \mathsf{vk}_0), (\mathcal{P}_1, \mathsf{vk}_1)$: The sharing algorithm outputs quantum program-shares $\mathcal{P}_0, \mathcal{P}_1$ encoding an input program $P \in \mathcal{C}$. It also outputs the corresponding classical verification keys $\mathsf{vk}_0, \mathsf{vk}_1$.

$\mathcal{Eval}(i, \mathcal{P}_i, x) \to y_i$: The evaluation algorithm takes an index $i \in \{0, 1\}$, a quantum program-share $\mathcal{P}_i$, and a classical input x. It outputs a classical evaluation-share y_i.

$\mathcal{Del}(i, \mathcal{P}_i) \to \mathsf{cert}_i$: The deletion algorithm takes an index $i \in \{0, 1\}$, a corresponding program-share $\mathcal{P}_i$, and produces a deletion certificate cert_i.

$\mathsf{Vrfy}(i, \mathsf{vk}_i, \mathsf{cert}_i) \to \top/\bot$: The classical verification algorithm takes an index $i \in \{0, 1\}$, the corresponding verification key vk_i and a certificate cert_i. It outputs $\top$ or $\bot$.

Deletion Correctness: The following condition holds for each index $i \in \{0, 1\}$:

$$\Pr\left[\mathsf{Vrfy}(i, \mathsf{vk}_i, \mathsf{cert}_i) \to \top \ : \ \begin{array}{l} (\mathcal{P}_0, \mathsf{vk}_0), (\mathcal{P}_1, \mathsf{vk}_1) \leftarrow \mathcal{Share}(P) \\ \mathsf{cert}_i \leftarrow \mathcal{Del}(i, \mathcal{P}_i) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

Evaluation Correctness: The following holds for every input x in the domain of P:

$$\Pr\left[y_0 \oplus y_1 = P(x) \ : \ \begin{array}{l} (\mathcal{P}_0, \mathsf{vk}_0), (\mathcal{P}_1, \mathsf{vk}_1) \leftarrow \mathcal{Share}(P) \\ y_i \leftarrow \mathcal{Eval}(i, \mathcal{P}_i, x) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

Remark 3.1. Notice that a correct y_i can be obtained from $\mathcal{P}_i$ with overwhelming probability. Hence, from the gentle measurement lemma, such a y_i can be obtained without almost disturbing the state. This implies that an arbitrary polynomial number of evaluations can be performed using $\mathcal{P}_i$.

3.2 Security Definitions

Definition 3.2 (Finite-Term Security). *The following security notion is defined wrt a non-local QPT $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, and a distribution $D_{\mathcal{C}}$ over $\mathcal{C}$:*

$\mathsf{Exp}^{\mathsf{fin\text{-}lessor}}_{\mathsf{DSL},\mathcal{A}}(1^\lambda, D_{\mathcal{C}})$**:**

1. *The challenger $\mathcal{C}h$ samples $P \leftarrow D_{\mathcal{C}}$, computes $(\mathcal{P}_0, \mathsf{vk}_0), (\mathcal{P}_1, \mathsf{vk}_1) \leftarrow \mathcal{S}hare(P)$ and sends $\mathcal{P}_0, \mathcal{P}_1$ to $\mathcal{A}_0, \mathcal{A}_1$ respectively.*
2. *$\mathcal{C}h$ receives $(\mathsf{cert}_0, \rho_0), (\mathsf{cert}_1, \rho_1)$ from $\mathcal{A}_0, \mathcal{A}_1$ respectively.*
3. *If $\mathsf{Vrfy}(i, \mathsf{vk}_i, \mathsf{cert}_i) = \top$ for some $i \in \{0,1\}$, $\mathcal{C}h$ samples $x^* \leftarrow \{0,1\}^\lambda$ and sends (x^*, ρ_0, ρ_1) to $\mathcal{A}_2$. Otherwise, it outputs $\perp$.*
4. *$\mathcal{A}_2$ sends y^* to $\mathcal{C}h$. If $y^* = P(x^*)$, $\mathcal{C}h$ outputs $\top$. Else, it outputs $\perp$.*

A DSL scheme DSL has deletion security if for all QPT adversaries $\mathcal{A}$, the following holds:

$$\Pr\left[\mathsf{Exp}^{\mathsf{fin\text{-}lessor}}_{\mathsf{DSL},\mathcal{A}}(1^\lambda, D_{\mathcal{C}}) \to \top\right] \leq \mathsf{negl}(\lambda)$$

Definition 3.3 (Secrecy). *The following security notion is defined wrt a QPT adversary $\mathcal{A}$:*

$\mathsf{Expt}^{\mathsf{ind}}_{\mathsf{DSL},\mathcal{A}}(1^\lambda, b)$**:**

1. *$\mathcal{A}$ sends (P^0, P^1) and $i \in \{0,1\}$ to $\mathcal{C}h$. $\mathcal{C}h$ runs $(\mathcal{P}_0, \mathsf{vk}_0), (\mathcal{P}_1, \mathsf{vk}_1) \leftarrow \mathcal{S}hare(P^b)$ and sends $\mathcal{P}_i$ to $\mathcal{A}$.*
2. *$\mathcal{A}$ sends b' to $\mathcal{C}h$. $\mathcal{C}h$ outputs b' as the final output.*

$$\left|\Pr\left[\mathsf{Expt}^{\mathsf{ind}}_{\mathsf{DSL},\mathcal{A}}(1^\lambda, 0) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{ind}}_{\mathsf{DSL},\mathcal{A}}(1^\lambda, 1) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

4 Inapplicability of SSL Impossibility

We will now briefly review the impossibility result regarding SSL by Ananth and LaPlaca [AL21]. They showed the existence of a class of unlearnable circuits that cannot be securely-leased (and thereby copy-protected). The corresponding circuit class $\mathcal{C}$ consists of circuits of the form $C_{a,b,r,\mathsf{pk},\mathcal{O}}(x)$, described as follows:

$\underline{C_{a,b,r,\mathsf{pk},\mathcal{O}}(x)\text{:}}$

1. If $x = 0 \dots 0$, output $\mathsf{QFHE}.\mathcal{E}nc(\mathsf{pk}, a; r) \| \mathcal{O} \| \mathsf{pk}$.
2. Else, if $x = a$, output b.
3. Otherwise, output $0 \dots 0$.

The corresponding description $D_{\mathcal{C}}$ samples a circuit C from $\mathcal{C}$ as follows:

$\underline{D_{\mathcal{C}}\text{:}}$

1. Sample $a, b, r \leftarrow \{0,1\}^\lambda$.
2. Compute $(\mathsf{pk}, \mathsf{sk}, \rho_{\mathsf{evk}}) \leftarrow \mathsf{QFHE}.\mathcal{G}en(1^\lambda)$.
3. Compute $\mathcal{O} \leftarrow \mathsf{Obf}\big(\mathsf{QFHE}.\mathcal{D}ec(\mathsf{sk}, \cdot), b, \mathsf{sk}\|r\big)$.
4. Output $C_{a,b,r,\mathsf{pk},\mathcal{O}}$.

It was shown in [AL21] that given (quantum) black-box access to C for $C \leftarrow D_C$, it is infeasible to learn the description of the circuit. However, given access to a quantum implementation (U_C, ρ_C) of $C \in \mathcal{C}$, there exists an algorithm $\mathcal{B}$ that learns the description of C. Since this description is classical, it can be copied. The algorithm $\mathcal{B}$ is described as follows:

$\underline{\mathcal{B}(U_C, \rho_C)}$**:**

1. Compute $(\rho', \mathsf{ct}_1 \| \mathcal{O}' \| \mathsf{pk}') \leftarrow U_C(\rho_C, 0 \ldots 0)$.
2. Compute $\sigma \| \mathsf{ct}_2 \leftarrow \mathsf{QFHE.Eval}(U_C(\rho', \cdot), \mathsf{ct}_1)$.
3. Compute $\mathsf{sk}' \| r \leftarrow \mathcal{O}(\mathsf{ct}_2)$.
4. $a' \leftarrow \mathsf{QFHE.Dec}(\mathsf{sk}', \mathsf{ct}_1), b' \leftarrow \mathsf{QFHE.Dec}(\mathsf{sk}', \mathsf{ct}_2)$.
5. Compute $a' \leftarrow \mathsf{QFHE.Dec}(\mathsf{sk}', \mathsf{ct}_1), b' \leftarrow \mathsf{QFHE.Dec}(\mathsf{sk}', \mathsf{ct}_2)$.
6. Output $C_{a',b',r',\mathsf{pk}',\mathcal{O}'}$.

It is trivially true that $\mathsf{pk}' = \mathsf{pk}$ and $\mathcal{O}' = \mathcal{O}$. Let us observe why $\mathcal{B}$ succeeds in obtaining $a' = a$ and $b' = b$. After Step 1., the state ρ' is close to the state ρ_C by correctness of evaluation. From the evaluation correctness of QFHE, ct_2 is a valid encryption of b. Therefore, $\mathsf{sk}' = \mathsf{sk}$ is output by the lockable obfuscation circuit $\mathcal{O}$. Finally, sk is used to decrypt $\mathsf{ct}_1, \mathsf{ct}_2$ to obtain $a' = a$ and $b' = b$ respectively. Furthermore, it was shown in their work that σ can be used to obtain a state ρ'' close to the state ρ_C which can then be deleted.

Observe now that an attack of the above kind does not work in the distributed setting. This is because a quantum share $\mathcal{P}_i$ of a circuit $P \in \mathcal{C}$ would only reveal an additive share of the FHE ciphertext ct_1 in Step 1. Hence, it is unclear if the description of $\mathcal{P}_i$ (or P) can be learnt before deletion, even in a way that the descriptions are only revealed after collusion.

Upon looking into it further, it seems like it should be impossible, at least for Encrypted Software Leasing (ESL). If the adversary could do a malleability attack, then that is enough. To show such an attack always exists is much harder than in the nice QFHE case though. One thing is that a malleability attack should exist (which is a classical crypto question) but we also need the fact that the attack doesn't disturb the state.

5 DSL for Puncturable Functions

- Try to construct from dIO-CD + PKE (similar to FSS construction).
- For the dIO-CD + PKE construction, we need to have a distributed diO. $\mathcal{A}_0$ receives one copy, $\mathcal{A}_1$ receives another (note that we don't care about the secret keys $\mathsf{sk}_0, \mathsf{sk}_1$ here as we can assume both have both unlike in the construction). $\mathcal{A}_0$ deletes and $\mathcal{A}_1$ deletes and they combine, so it is even weaker than 2-collusion-resistant security.

6 Distributed One-Time Programs

One time programs allow a software to be encoded into a format that enables only a single evaluation. Unfortunately, deterministic programs cannot be one-time protected. In this work, we explore the setting where a deterministic program can be split between two parties. In this setting, we explore whether it is possible to enforce only a single distributed evaluation. We show a feasibility where the program shares are entangled and prove that such entanglement is necessary.

Use Case: A party can distribute a program to two servers who are supposed to execute the software within a short period. For proof, they can post commitments to their shares online or on the blockchain. During this time, one might be able to enforce that the servers do not directly collude with each other due to the fear of apprehension. However, they might eventually sell their data to random entities on the black market. This notion guarantees that if the initial evaluation was correctly performed in a distributed fashion, nothing can be done to evaluate a second time.

6.1 Definition

A Distributed One-Time Program (D-OTP) scheme $\mathsf{DOTP} = (\mathsf{KG}, \mathcal{TG}, \mathcal{TE})$ has the following syntax:

Syntax:

$\mathsf{KG}(f) \to (\widetilde{P}_0, \widetilde{P}_1, \mathsf{sk})$: The key-generation algorithm outputs two classical programs $\widetilde{P}_0, \widetilde{P}_1$ and a secret-key sk.

$\mathcal{TG}(\mathsf{sk}) \to (\mathcal{tk}_0, \mathcal{tk}_1)$: The token-generation algorithm takes in a secret-key and generates (possibly-entangled) quantum tokens $\mathcal{tk}_0, \mathcal{tk}_1$.

$\mathcal{TE}(x, \mathcal{tk}_i, \widetilde{P}_i) \to y_i$: The token-evaluation algorithm takes a token $\mathcal{tk}_i$, a program $\widetilde{P}_i$, and a classical input x. It outputs a classical evaluation-share y_i.

Evaluation Correctness: The following holds for every input x in the domain of f:

$$\Pr\left[y_0 \oplus y_1 = f(x) \;:\; \begin{array}{l} (\widetilde{P}_0, \widetilde{P}_1, \mathsf{sk}) \leftarrow \mathsf{KG}(f) \\ (\mathcal{tk}_0, \mathcal{tk}_1) \leftarrow \mathcal{TG}(\mathsf{sk}) \\ \forall i \in \{0,1\} : y_i \leftarrow \mathcal{TE}(x, \mathcal{tk}_i, \widetilde{P}_i) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda)$$

Remark 6.1. Due to the gentle-measurement lemma, the evaluation correctness implies that the state described by $\mathcal{tk}_0, \mathcal{tk}_1$ can be evaluated on without almost disturbing the state. This seems to contradict any meaningful notion of one-time security. However, our distributed notion will make use of the fact that the non-local adversary must produce a correct distributed evaluation first before collusion is allowed. Since the tokens $\mathcal{tk}_0, \mathcal{tk}_1$ can be entangled, y_0 and y_1 can be truly random in a local-sense (even for a fixed global token state).

Definition 6.2 (Distributed One-Time Security). *Consider the following experiment between a challenger $\mathcal{Ch}$ and a QPT non-local adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ for the scheme* $\mathsf{DOTP} = (\mathsf{KG}, \mathcal{TG}, \mathcal{TE})$ *and a function f sampled from a distribution $\mathcal{D}$:*

$\underline{\mathsf{Exp}^{\text{d-ots}}_{\mathsf{DOTP},\mathcal{A}}(1^\lambda, f, \mathcal{D})}$*:*

1. *$\mathcal{Ch}$ generates $f \leftarrow \mathcal{D}$, $(\widetilde{P}_0, \widetilde{P}_1, \mathsf{sk}) \leftarrow \mathsf{KG}(f)$ and $(\mathcal{tk}_0, \mathcal{tk}_1) \leftarrow \mathcal{TG}(\mathsf{sk})$.*
2. *$\mathcal{Ch}$ sends $(\mathcal{tk}_i, \widetilde{P}_i)$ to adversary $\mathcal{A}_i$ for each $i \in \{0,1\}$.*
3. *For $i \in \{0,1\}$, if adversary $\mathcal{A}_i$ sends a register R_i, run $\widetilde{P}_i$ on it and measure the output register to get outcome y_i. Send the register R_i to adversary $\mathcal{A}_2$.*
4. *Define $y := y_0 \oplus y_1$.*
5. *For each $i \in \{0,1\}$, wait to receive register R'_i from $\mathcal{A}_2$. Then, run $\widetilde{P}_i$ on it and measure the output register to get outcome y'_i.*
6. *Define $y' := y'_0 \oplus y'_1$.*
7. *If $y' \neq y$, output $\top$. Else, output $\bot$.*

We require the following condition to hold:

$$\Pr\left[\mathsf{Exp}^{\text{d-ots}}_{\mathsf{DOTP},\mathcal{A}}(1^\lambda, f, \mathcal{D}) \to \top\right] \leq \mathsf{negl}(\lambda)$$

Notice that the above definition allows collusion after the first distributed evaluation. As per the definition, f can even be a deterministic function. It remains to be seen which functions can be handled. At the very least, f must be unlearnable given a single black-box query, as the “collusion-stage" adversary $\mathcal{A}_2$ can compute the first evaluation y. The quintessential example would be a PRF family $\{F_k\}_k$, i.e. $P = F_k$ for $k \leftarrow \{0,1\}^\lambda$ which is clearly unlearnable.

6.2 Local One-Time Authentication

A Local One-Time Authentication (LOTA) scheme is a tuple of 5 algorithms $\mathsf{LOTA} = (\mathsf{KG}, \mathcal{TG}, \mathcal{Sign}, \mathsf{Vrfy})$ with the following syntax:

$\mathsf{KG}(1^\lambda) \to \mathsf{sk}$: The key-generation algorithm outputs a classical secret-key sk.

$\mathcal{TG}(\mathsf{sk}) \to (\mathcal{tk}_0, \mathcal{tk}_1)$: The token-generation algorithm takes the secret-key sk as input. It outputs (possibly-entangled) quantum token states $(\mathcal{tk}_0, \mathcal{tk}_1)$.

$\mathcal{Sign}(x, \mathcal{tk}_i) \to z$: The signing algorithm takes as input a token state $\mathcal{tk}_i$ and a message x. It outputs a signature z.

$\mathsf{Vrfy}(\mathsf{sk}, (x, z)) \to \top/\bot$ The verification algorithm Vrfy takes as input the secret-key sk and a message-signature pair (x, z). It outputs $\top$ or $\bot$.

Same Signatures: The following holds for all $x \in \{0,1\}$:

$$\Pr\left[z_0 = z_1 \ : \ \begin{array}{l} \mathsf{sk} \leftarrow \mathsf{KG}(1^\lambda) \\ (\mathcal{tk}_0, \mathcal{tk}_1) \leftarrow \mathcal{TG}(\mathsf{sk}) \\ z_0 \leftarrow \mathcal{Sign}(x, \mathcal{tk}_0) \\ z_1 \leftarrow \mathcal{Sign}(x, \mathcal{tk}_1) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda)$$

Verification Correctness: The following holds for all $x \in \{0,1\}$ and $i \in \{0,1\}$:

$$\Pr\left[\mathsf{Vrfy}(\mathsf{sk}, (x, z)) \to \top \ : \ \begin{array}{l} \mathsf{sk} \leftarrow \mathsf{KG}(1^\lambda) \\ (\mathcal{tk}_0, \mathcal{tk}_1) \leftarrow \mathcal{TG}(\mathsf{sk}) \\ z \leftarrow \mathcal{Sign}(x, \mathcal{tk}_i) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda)$$

Definition 6.3 (One-Time Authentication). *Consider the following experiment $\mathsf{Expt}^{\mathsf{ota}}_{\mathsf{LOTA},\mathcal{A}}(1^\lambda)$ between a challenger $\mathcal{Ch}$ and a QPT adversary $\mathcal{A}$:*

$\underline{\mathsf{Expt}^{\mathsf{ota}}_{\mathsf{LOTA},\mathcal{A}}(1^\lambda)}$*:*

1. *$\mathcal{Ch}$ runs $\mathsf{sk} \leftarrow \mathsf{KG}(1^\lambda)$ and $(\mathcal{tk}_0, \mathcal{tk}_1) \leftarrow \mathcal{TG}(\mathsf{sk})$.*
2. *$\mathcal{Ch}$ sends $\mathcal{tk}_0$ to $\mathcal{A}$.*
3. *$\mathcal{A}$ sends $(z, \widetilde{z})$ to $\mathcal{Ch}$.*
4. *If $\mathsf{Vrfy}(\mathsf{sk}, (0, z)) = \mathsf{Vrfy}(\mathsf{sk}, (1, \widetilde{z})) = \top$, $\mathcal{Ch}$ outputs $\top$. Otherwise, it output $\bot$.*

We require that the following condition holds:

$$\Pr\left[\mathsf{Expt}^{\mathsf{ota}}_{\mathsf{LOTA},\mathcal{A}}(1^\lambda) = \top\right] \leq \mathsf{negl}(\lambda)$$

6.3 Construction in the Oracle Model

We will use the following idealized oracles:

- Random Oracle H.
- VBB Obfuscation Obf.
- Local One-Time Authentication Scheme $\mathsf{LOTA} = \mathsf{LOTA}.(\mathsf{KG}, \mathcal{TG}, \mathcal{Sign}, \mathsf{Vrfy})$.

```
$\underline{\mathsf{KG}(f)}$ :

    • Sample $\mathsf{sk} \leftarrow \mathsf{LOTA.KG}(1^\lambda)$.
    • Let $P$ be the following program and $\widetilde{P} := \mathsf{Obf}(P)$.
      $\underline{P_0(x,z)}$ :
          – If $\mathsf{Vrfy}(\mathsf{sk}, (x,z)) = \bot$, output $\bot$.
          – Otherwise, output $P(x) \oplus H(z)$.
    • Output $(\widetilde{P}_0, \widetilde{P}_1, \mathsf{sk})$.

$\underline{\mathcal{TG}(\mathsf{sk})}$ :

    • Compute $(t\mathcal{K}_0, t\mathcal{K}_1) \leftarrow \mathsf{LOTA}.\mathcal{TG}(\mathsf{sk})$.
    • Output $(t\mathcal{K}_0, t\mathcal{K}_1)$.

$\underline{\mathcal{TE}(x, t\mathcal{K}_i, \widetilde{P}_i)}$ :

    • Compute $z \leftarrow \mathcal{S}ign(x, t\mathcal{K}_i)$.
    • Compute and output $y_i := \widetilde{P}_i(x,z)$.
```

References

[AL21] Prabhanjan Ananth and Rolando L. La Placa. Secure software leasing. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 501–530. Springer, Heidelberg, October 2021. (Cited on page 3, 5, 6.)