Asymmetric Trapdoor Pseudorandom Generators: Definitions, Constructions, and Applications to Homomorphic Signatures with Shorter Public Keys *

Jinpeng Hou[1], Yansong Gao[2], Mang Su[3], Willy Susilo[4], Jie Chen[5], and Anmin Fu[1,3] (✉)

[1] School of Cyber Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China
{jinpenghou, fuam}@njust.edu.cn
[2] Data61, CSIRO, Sydney, Australia
garrison.gao@data61.csiro.au
[3] School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China
sumang@njust.edu.cn
[4] School of Computing and Information Technology, Institute of Cybersecurity and Cryptology, University of Wollongong, Wollongong, NSW 2522, Australia
wsusilo@uow.edu
[5] School of Software Engineering, East China Normal University, Shanghai 200062, China
s080001@e.ntu.edu.sg

Abstract. We introduce a new primitive called the asymmetric trapdoor pseudorandom generator (ATPRG), which belongs to pseudorandom generators with two additional trapdoors (a public trapdoor and a secret trapdoor) or backdoor pseudorandom generators with an additional trapdoor (a secret trapdoor). Specifically, ATPRG can only generate public pseudorandom numbers $pr_1, \ldots, pr_n$ for the users having no knowledge of the public trapdoor and the secret trapdoor; so that this function is the same as pseudorandom generators. However, the users having the public trapdoor can use any public pseudorandom number pr_i to recover the whole pr sequence; so that this function is the same as backdoor pseudorandom generators. Further, the users having the secret trapdoor can use pr sequence to generate a sequence $sr_1, \ldots, sr_n$ of the secret pseudorandom numbers.

As for applications of ATPRG, we construct the first homomorphic signature scheme (in the standard model) whose public key size is only $O(T)$ independent of the dataset size. As a comparison, the shortest size of the existing public key is $O(\sqrt{N} + \sqrt{T})$, proposed by Catalano et al. (CRYPTO'15), where N is the dataset size and T is the dimension of

* Corresponding author: Anmin Fu. This work is supported by National Natural Science Foundation of China (62072239, 61972094), and Natural Science Foundation of Jiangsu Province, China (BK20211192).

the message. In other words, we provide the first homomorphic signature scheme with $O(1)$-sized public keys for the one-dimension messages.

Keywords: Pseudorandom generators · Homomorphic MAC · Homoprhic signatures · Standard model.

1 Introduction

1.1 Background

Some primitives of public key cryptography, such as multi-key fully homomorphic encryption and homomorphic signatures (HS), have one main limitation that the public key size is linear with the dataset size. When the dataset is large, the public key size may reach MB or even GB magnitude, and the resulting storage and communication overheads are burdensome for resource-constrained devices, e.g., mobile devices and Internet of Thing (IoT) devices. To the best of our knowledge, only Catalano et al. (CRYPTO'15) [7] proposed a HS scheme in which the public key size is less than $O(N+T)$ in the past two decades, where N is the dataset size and T is the dimension of the message.

HS schemes [20] allow a remote party to compute any functions from a class of admissible functions over signed messages and derive verifiable signatures given computed results, which can verify the correctness of the corresponding computed results efficiently. HS can solve many problems in cloud computing, such as data integrity checking, verifiable computation, electronic voting [17]. According to the identity of the verifier, HS can divide into homomorphic MAC (HA) and signatures. The difference between the two is that the former is privately verified, and the latter can be verified publicly. In the past two decades, HS has undergone great development in the computed function, especially from linearly HS that only supports addition or multiplication [1,7,9,10,14,15,18,20,22,23,27], to bounded polynomials [2, 3, 5, 8] and to (level) fully homomorphic operations that allow computing the general arithmetic circuits of a priori bounded depth [13, 16, 17]. However, as a notable limitation, in the standard model, the public key sizes of [1–3,5,8–10,13–15,17,18,20,22,23,27] are all restricted to be linear with the dataset size; the smallest of [7] is $O(\sqrt{N}+\sqrt{T})$.

It is challenging to reduce the public key size, as there is no existing universal technique. In this paper, we attempt to design a universal technique to shorten public keys from the perspective of pseudorandom number generators (PRGs). In cryptographic theory, the security of most cryptographic tasks critically depends on the randomness quality. Since true random bits are hard to generate without specialized hardware, PRGs are often alternatively used in cryptographic schemes. PRG takes a short random seed as input and outputs the bit-strings of arbitrary (polynomial) length. We can divide it into two categories: non-backdoor PRG and backdoor PRG (BPRG) [11,12,28], according to whether a backdoor exists. Non-backdoor PRG can only output pseudorandom numbers. BPRG is a standard PRG for adversary $\mathscr{A}$ without the backdoor $\mathsf{td}_{\mathscr{B}}$;

however, there also exists another adversary $\mathscr{B}$ in BPRG who has the backdoor $\mathsf{td}_{\mathscr{B}}$ [11]. $\mathscr{B}$ can use the backdoor and any BPRG output (a pseudorandom number) to recover all the forward and backward pseudorandom numbers of the BPRG output.

At the high level, the public keys are just a string of pseudorandom numbers, and if we can compress n pseudorandom numbers into a shorter string, it appears that we can shorten length of the public keys. BPRG can provide similar functionality since $\mathscr{B}$ can use the backdoor to recover the whole sequence of the pseudorandom numbers. Unfortunately, the public keys only appear as pseudorandom numbers but have some mappings with the secret keys (e.g., $\phi_1 : sk \to pk$ is computationally easy, but $\phi_1^{-1} : pk \to sk$ is computationally hard). Therefore, the function of the public keys cannot be completely replaced by the pseudorandom numbers generated by BPRG. We need a new security primitive for generating the public keys so that it not only has the function of BPRG but also can use the generated pseudorandom numbers (public keys) to provision the corresponding secret keys. The above idea may sound like an intuitive way to break the computationally hard condition of the ϕ_1^{-1} map; counter-intuitively, we use an unusual bilinear group to build this new primitive, as will be described later.

1.2 Our Contributions

ATPRG: We introduce a new primitive that we call the asymmetric trapdoor pseudorandom generator (ATPRG). Then we construct an ATPRG instance upon techniques of PRG, reverse re-randomizable encryption, and trapdoor projection maps for bilinear groups. ATPRG has five algorithms ($\mathsf{setup}, \mathsf{init}, \mathsf{PRGen}, \mathsf{SRGen}, \mathsf{rec}_{\mathscr{B}}$), where PRGen and SRGen are two algorithms for generating pseudorandom numbers, and setup can generate a public trapdoor $\mathsf{td}_{\mathscr{B}}$ and a secret trapdoor $\mathsf{td}_{\mathscr{S}}$. The users without $\mathsf{td}_{\mathscr{B}}$ and $\mathsf{td}_{\mathscr{S}}$ can only use the three algorithms ($\mathsf{setup}, \mathsf{init}, \mathsf{PRGen}$), which are equivalent to the non-backdoor PRG. The users only have $\mathsf{td}_{\mathscr{B}}$ can use the four algorithms ($\mathsf{setup}, \mathsf{init}, \mathsf{PRGen}, \mathsf{rec}_{\mathscr{B}}$), which are equivalent to BPRG. The users with $\mathsf{td}_{\mathscr{B}}$ and $\mathsf{td}_{\mathscr{S}}$ can run the whole five algorithms.

More specifically, PRGen produces a sequence $\overline{pr} = \{pr_1, \ldots, pr_n\}$ of public pseudorandom numbers, and SRGen (using $\overline{pr}$ and the secret trapdoor $\mathsf{td}_{\mathscr{S}}$) produces a corresponding sequence $\overline{sr} = \{sr_1, \ldots, sr_n\}$ of secret pseudorandom numbers. $\overline{pr}$ is to generate the public keys, and $\overline{sr}$ is to generate the corresponding secret keys. We require the following relationships to hold among these four sequences: $\phi_1 : sk \to pk$, $\phi_2 : pr(\mathsf{td}_{\mathscr{S}}) \to sr$, and $\phi_3 : sr \to sk$ are computationally easy, but $\phi_1^{-1} : pk \to sk$ and $\phi_4 : pr \to sr$ are computationally hard. In addition, ($\mathsf{setup}, \mathsf{init}, \mathsf{PRGen}, \mathsf{rec}_{\mathscr{B}}$) constitutes a BPRG; that is, one with a backdoor $\mathsf{td}_{\mathscr{B}}$ (public trapdoor) that can calculate all outputs of $\overline{pr}$ by algorithm $\mathsf{rec}_{\mathscr{B}}$ using any PRGen output pr_i. Therefore, the properties of these five mappings guarantee the confidentiality of the secret keys and the compressibility of the public keys.

The challenge of designing ATPRG is *how to construct the secret trapdoor* $\mathsf{td}_{\mathscr{S}}$. We address $\mathsf{td}_{\mathscr{S}}$ construction by a trapdoor projection map for bilinear groups. We first convert pr_i to a random element pr'_i of elliptic curve group G and then map pr'_i to a torsional subgroup G_1 of G by the trapdoor projection map. The resulting image of this mapping is the corresponding secret pseudorandom number sr_i (random element of group G_1). According to the subgroup hiding hypothesis, $\overline{sr}$ cannot be calculated effectively from $\overline{pr}$ without knowing the secret trapdoor $\mathsf{td}_{\mathscr{S}}$. ATPRG provides a universal method to shorten the public key size, which allows any user to recover the sequence of public keys of any size through the public trapdoor $\mathsf{td}_{\mathscr{B}}$ with a fixed size. At the same time, the secret trapdoor $\mathsf{td}_{\mathscr{S}}$ is used to ensure the confidentiality of secret keys. In other words, ATPRG can reduce the storage and communication overheads of the public keys of arbitrary length to a shortened size (e.g., $O(1)$) for some cryptographic schemes.

Application: For the HS schemes (in the standard model), the public key size is generally larger than or equal to $O(N+T)$, where N is the dataset size and T is the dimension of the message. To the best of our knowledge, only [7] proposed a HS scheme with $O(\sqrt{N}+\sqrt{T})$-sized public keys exploiting techniques of programmable hash and cover-free. We use ATPRG to design the first HS scheme (in the standard model) whose public key size is independent of the dataset size, and our scheme has only $O(T)$-sized public keys. Specifically, for n T-dimension messages, we use PRGen algorithm to generate n pseudorandom numbers $R_1, \ldots, R_n$, then use SRGen algorithm to transform $R_1, \ldots, R_n$ to $S_1, \ldots, S_n$. A simplified structure of our signature is as following:

$$\left(S_i \cdot \prod_{j=1}^{T} h_j^{m_i[j]} \cdot W_i \right)^{\alpha},$$

where $m_i[j]$ represents the j-th component of the i-th message, α is secret key, h_j and W_i are pseudorandom numbers, $h_1, \ldots, h_T$ are generated in the key generation step, and W_i is generated by the signer. For a computed result $\overline{m} = \prod_{i=1}^{n} c_i * m_i$ (c_i are coefficients of linear functions), we can verify the correctness of $\overline{m}$ by using $R_1, \ldots, R_n$ through an unusual bilinear mapping. In our HS scheme, $R_1, \ldots, R_n$ and $h_1, \ldots, h_T$ are the public keys, we only need to store one element R_i for any $i \in [n]$ since the algorithm $\mathsf{rec}_{\mathscr{B}}$ can use R_i and public trapdoor $\mathsf{td}_{\mathscr{B}}$ to recover the whole sequence $R_1, \ldots, R_n$. Admittedly, the PRGen algorithm of our ATPRG instance is not fine-grained. If we use it to generate $h_1, \ldots, h_T$, although we can get a HS scheme with $O(1)$-sized public keys for the T-dimension messages, it would cause trouble for security reduction. Therefore, our final HS scheme has the $O(T)$-sized public keys for the T-dimension messages. The notable benefit of this construction is that we can prove the unforgeability of our HS scheme based on the discrete logarithm (DL) and computational Diffie-Hellman (CDH) assumptions in the standard model. We summarized and compare the public key sizes of different schemes in table 1.

ATPRG can help design more space-efficient protocols where data/input/message should respect a predefined (unchangeable) order to be correctly processed in a computation or malleable cryptographic system. We believe that the shortest HS scheme ($O(1)$-sized public keys for the T-dimension messages) can be obtained asymptotically by this new primitive.

Table 1: Comparison of public key size (in the standard model)

Schemes	Public keys	Secret keys
[1–3, 5, 8–10, 13–15, 17, 18, 20, 22, 23, 27]	$O(N+T)$	$O(1) \sim O(N+T)$
[7]	$O(\sqrt{N}+\sqrt{T})$	$O(\sqrt{N}+\sqrt{T})$
Our scheme	$O(T)$	$O(1)$

1.3 Organization

In Section 2, we introduce necessary notations and definitions, which will be utilized to construct the proposed ATPRG. In Section 3, we give the definitions of ATPRG and construct an ATPRG instance upon techniques of PRG, reverse re-randomizable encryption, and trapdoor projection maps for bilinear groups. In Section 4, we elaborate on the construction of our HS scheme with shorter public keys. We conclude this paper and discuss other ATPRG applications in Section 5.

2 Preliminaries

2.1 Notation

Let λ denote a security parameter, $a \xleftarrow{\$} A$ denotes that sampling uniformly at random the value a from the distribution A. $\eta(\lambda)$ represents a class of negligible function on λ. $[q]$ represents $1, \ldots, q$. The logarithms in this paper are to base two.

2.2 Reverse Re-randomizable Encryption

Before introducing the reverse re-randomizable encryption scheme, we first give two definitions of IND$-CPA-secure PKE and re-randomizable encryption since the latter two are the basis for building the former.

Definition 1. *A (t, q, δ)-IND$-CPA-secure PKE scheme [12] has three Probabilistic polynomial time (PPT) algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ such that for all adversaries $\mathscr{A}$ running in time t and making at most q queries, it holds that*

$$\mathsf{Adv}_{\mathscr{A}}^{\mathrm{IND\$-CPA}} := |\mathsf{Pr}[(pk, sk) \leftarrow \mathsf{KeyGen} : \mathscr{A}^{\mathsf{Enc}(pk,\cdot)}(pk) = 1] - \mathsf{Pr}[(pk, sk) \leftarrow \mathsf{KeyGen} : \mathscr{A}^{\$(\cdot)}(pk) = 1]| \leq \delta,$$

where $\$(\cdot)$ *is such that on input a message* M*, it returns a random string of size* $|\mathsf{Enc}(pk, M)|$*.*

Apparently, a (t, q, δ)-IND$-CPA-secure PKE scheme is also a $(t, q, 2\delta)$-IND-CPA-secure PKE scheme. Since the backdoor pseudorandom generators, re-randomizable encryption, and reverse re-randomizable encryption in this paper all use pseudorandom ciphertext, we focus on the IND$-CPA-secure PKE scheme.

Definition 2. *A* (t, q, δ, ν)-IND$-CPA*-secure re-randomizable encryption scheme [19] has four PPT algorithms* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand})$*, where* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is a standard* IND$-CPA*-secure PKE scheme (which implies the correctness of PKE).* Rand *is an efficiently randomised algorithm such that for all* $(pk, sk) \leftarrow \mathsf{KeyGen}$*,* M*, and* R_1*, we have*

$$\Delta(\left\{R_0 \xleftarrow{\$} \mathsf{Coins}(\mathsf{Enc}) : \mathsf{Enc}(pk, M; R_0)\right\},$$
$$\left\{R' \xleftarrow{\$} \mathsf{Coins}(\mathsf{Rand}) : \mathsf{Rand}(\mathsf{Enc}(pk, M; R_1); R')\right\}) \leq \nu,$$

where $\mathsf{Coins}(A)$ *denotes the distribution of the internal randomness of* A *so that the distribution* $\left\{a \xleftarrow{\$} A\right\}$ *is actually* $\left\{r \xleftarrow{\$} \mathsf{Coins}(A) : a = A(r)\right\}$*. That is, the distributions of a ciphertext generated by a standard PKE scheme and a ciphertext generated by* Rand *with arbitrary randomness are statistically close.*

The reverse re-randomizable encryption [11] scheme has an additional property compared to the re-randomizable encryption scheme, which is reverse re-randomizable.

Definition 3. *A* (t, q, δ, ν)-IND$-CPA*-secure reverse re-randomizable encryption scheme has five PPT algorithms* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand}, \mathsf{Rand}^{-1})$*, where* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand})$ *is a* (t, q, δ, ν)-IND$-CPA*-secure re-randomizable encryption scheme.* Rand^{-1} *is an efficient algorithm such that for all* $(pk, sk) \leftarrow \mathsf{KeyGen}$*,* M*,* R_0*, and* R_1*, we have*

$$\mathsf{Pr}[\mathsf{Rand}^{-1}(\mathsf{Rand}(\mathsf{Enc}(pk, M; R_0); R_1); R_1) = \mathsf{Enc}(pk, M; R_0)] = 1.$$

2.3 Pseudorandom Generators

A PRG takes a short random seed as input and outputs the bit-strings of arbitrary (polynomial) length. Following [11, 12], we also equip PRG with a parameter generation algorithm setup, which can simplify the following description of the backdoor PRG.

Definition 4. *A* PRG *has three PPT algorithms* $(\mathsf{setup}, \mathsf{init}, \mathsf{next})$ *which can be defined with two parameters* $(n, l) \in \mathbb{N}^2$*:*

- $(pp, bk) \leftarrow \mathsf{setup}(coins) : \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$*. This algorithm takes a random coin as input and outputs a public parameter* pp *and a secret backdoor parameter* bk*. For a non-backdoor PRG,* $bk = \perp$*.*

- $s \leftarrow \mathsf{init}(pp, coins) : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^n$. *This algorithm takes a public parameter pp and a random coin as input and outputs an initial state* $s_0 \in \{0,1\}^n$ *of PRG.*
- $(r, s') \leftarrow \mathsf{next}(pp, s) : \{0,1\}^* \times \{0,1\}^n \rightarrow \{0,1\}^l \times \{0,1\}^n$. *This algorithm takes a public parameter pp and a state* $s \in \{0,1\}^n$ *as input and outputs a pseudorandom l-bits string and a next state* s'.

According to the PRG definition, as long as we initialize a PRG, we can iterate the next algorithm to generate a fairly long pseudorandom bit string. Suppose that we iterate k next algorithm, we can let $\mathsf{out}^k(\mathsf{next}) = r_1, \ldots, r_k$ and $\mathsf{state}^k(\mathsf{next}) = s_1, \ldots, s_k$ to represent the k outputs of the pseudorandom numbers and the states. Next, we give some security definitions of PRG.

Definition 5 (PRG Distinguishing Advantage). *The distinguishing advantage of* $\mathscr{A}$ *against* PRG *is defined as follows:*

$$\mathsf{Adv}^{\mathsf{PRG.dist}}_{\mathscr{A},q} := 2 \left| \mathsf{Pr} \left[\begin{array}{c} (pp, bk) \leftarrow \mathsf{setup}(coins), \\ s_0 \leftarrow \mathsf{init}(pp, coins), \\ r_1^0, \ldots, r_q^0 \leftarrow \mathsf{out}^q(\mathsf{next}), \\ r_1^1, \ldots, r_q^1 \xleftarrow{\$} \{0,1\}^l, \\ b \xleftarrow{\$} \{0,1\}, \\ b' \leftarrow \mathscr{A}(pp, r_1^b, \ldots, r_q^b) : \\ b = b' \end{array} \right] - \frac{1}{2} \right|.$$

The adversary $\mathscr{A}$ *receives either q PRG outputs or q random l-bits strings.*

Based on the PRG distinguishing advantage, we can define a (t, q, δ)-secure PRG.

Definition 6 ((t, q, δ)-secure PRG). *A* PRG *is said to be* (t, q, δ)*-secure if for all adversaries* $\mathscr{A}$ *running in time t, it holds that* $\mathsf{Adv}^{\mathsf{PRG.dist}}_{\mathscr{A},q} \leq \delta$.

We can define a stronger security definition of PRG if we give the extra final state to the adversary $\mathscr{A}$, called forward security.

Definition 7 (PRG Forward-security Advantage). *The forward-security advantage of* $\mathscr{A}$ *against* PRG *is defined as follows:*

$$\mathsf{Adv}^{\mathsf{PRG.fwd}}_{\mathscr{A},q} := 2 \left| \mathsf{Pr} \left[\begin{array}{c} (pp, bk) \leftarrow \mathsf{setup}(coins), \\ s_0 \leftarrow \mathsf{init}(pp, coins), \\ r_1^0, \ldots, r_q^0 \leftarrow \mathsf{out}^q(\mathsf{next}), \\ r_1^1, \ldots, r_q^1 \xleftarrow{\$} \{0,1\}^l, \\ s_1, \ldots, s_q \leftarrow \mathsf{state}^q(\mathsf{next}), \\ b \xleftarrow{\$} \{0,1\}, \\ b' \leftarrow \mathscr{A}(pp, r_1^b, \ldots, r_q^b, s_q) : \\ b = b' \end{array} \right] - \frac{1}{2} \right|.$$

The adversary $\mathscr{A}$ *receives either q PRG outputs or q random l-bits strings.*

Definition 8 ((t, q, δ)-FWD-secure PRG). *A* PRG *is said to be* (t, q, δ)-FWD-*secure if for all adversaries* $\mathscr{A}$ *running in time t, it holds that* $\mathsf{Adv}^{\mathsf{PRG.fwd}}_{\mathscr{A},q} \leq \delta$.

2.4 Backdoor Pseudorandom Generators

Differing from the standard PRG, there is an another backdoor attacker $\mathscr{B}$ in BPRG; that is, there exists two attackers in a BPRG system, $\mathscr{A}$ and $\mathscr{B}$. The goal of $\mathscr{A}$ is to break the security of the BPRG scheme without access to the backdoor $\mathsf{td}_{\mathscr{B}}$, in this case, the security model of BPRG is the same as PRG. However, attacker $\mathscr{B}$ can recover the backward/forward outputs of BPRG using the backdoor $\mathsf{td}_{\mathscr{B}}$.

Definition 9. *A* BPRG *has four PPT algorithms* $(\mathsf{setup}, \mathsf{init}, \mathsf{next}, \mathsf{rec}_{\mathscr{B}})$*:*

- $(pp, bk) \leftarrow \mathsf{setup}(coins)$*. This algorithm takes a random coin as input and outputs a public parameter* pp *and a secret backdoor parameter* bk*.*
- $s_0 \leftarrow \mathsf{init}(pp, coins)$*. This algorithm takes a public parameter* pp *and a random coin as input and outputs an initial value* s_0 *of* BPRG*.*
- $(r, s') \leftarrow \mathsf{next}(pp, s)$*. This algorithm takes a public parameter* pp *and a* BPRG *state value* s *as input and outputs a random bit-string and a next* BPRG *state value* s'*. This step seems to be the same as PRG, but for the backdoor setting, the internal process differs from PRG.*
- $out_{\mathscr{B}} \leftarrow \mathsf{rec}_{\mathscr{B}}(bk, r_i, i, pp)$*. This algorithm inputs a backdoor* $bk = \mathsf{td}_{\mathscr{B}}$*, a pseudorandom number* r_i *generated by* BPRG.next *with its index* i*, and a public parameter* pp*. It outputs some forward or backward values* $out_{\mathscr{B}}$*, where* $out_{\mathscr{B}}$ *maybe the sequence of the pseudorandom numbers* $r_1, \ldots, r_i, \ldots$ *or the sequence of* BPRG *state values* $s_0, \ldots, s_{i-1}, \ldots$*.*

Following [12], BPRG has three security advantages for attacker $\mathscr{B}$: $\mathsf{Adv}^{\mathsf{BPRG.dist}}_{\mathscr{B},q}$, $\mathsf{Adv}^{\mathsf{BPRG.next}}_{\mathscr{B},q}$, and $\mathsf{Adv}^{\mathsf{BPRG.rsk}}_{\mathscr{B},q}$. And then [11] gives two stronger security advantages $\mathsf{Adv}^{\mathsf{BPRG.first}}_{\mathscr{B},q}$ and $\mathsf{Adv}^{\mathsf{BPRG.out}}_{\mathscr{B},q}$. We only give the definitions of $\mathsf{Adv}^{\mathsf{BPRG.first}}_{\mathscr{B},q}$ and $\mathsf{Adv}^{\mathsf{BPRG.out}}_{\mathscr{B},q}$ since they imply the correctness of our ATPRG to recover public pseudorandom sequence. The original paper of BPRG does not provide a direct definition of correctness; instead, it uses $\mathsf{Adv}^{\mathsf{BPRG.first}}_{\mathscr{B},q}$ and $\mathsf{Adv}^{\mathsf{BPRG.out}}_{\mathscr{B},q}$ to imply the correctness advantage, probably because the authors are targeting "Big Brother" to research.

Definition 10 (BPRG First Advantage). *The first advantage of* $\mathscr{B}$ *against* BPRG *is defined as follows:*

$$\mathsf{Adv}^{\mathsf{BPRG.first}}_{\mathscr{B},q} := \mathsf{Pr}\begin{bmatrix}(pp, bk) \leftarrow \mathsf{setup}(coins),\\ s_0 \leftarrow \mathsf{init}(pp, coins),\\ r_1, \ldots, r_q \leftarrow \mathsf{out}^q(\mathsf{next}),\\ s_0^* \leftarrow \mathscr{B}(pp, \mathsf{td}_{\mathscr{B}}, r_i) :\\ s_0 = s_0^*\end{bmatrix}.$$

Definition 11 (BPRG Output Advantage). *The output advantage of* $\mathscr{B}$ *against* BPRG *is defined as follows:*

$$\mathsf{Adv}^{\mathsf{BPRG.out}}_{\mathscr{B},q} := \mathsf{Pr}\left[\begin{array}{c}(pp, bk) \leftarrow \mathsf{setup}(coins),\\ s_0 \leftarrow \mathsf{init}(pp, coins),\\ r_1, \ldots, r_q \leftarrow \mathsf{out}^q(\mathsf{next}),\\ r_1^*, \ldots, r_q^* \leftarrow \mathscr{B}(pp, \mathsf{td}_{\mathscr{B}}, r_i, i) :\\ r_1, \ldots, r_q = r_1^*, \ldots, r_q^*\end{array}\right].$$

Based on the above five security advantages of BPRG, we can define a $(t, q, \delta, (\cdot, \epsilon))$ (-FWD)-secure BPRG, where the $\cdot$ in $(\cdot, \epsilon)$ represents dist/next/rsk/first/out.

Definition 12 ($(t, q, \delta, (\cdot, \epsilon))$-secure BPRG). *A* $\mathsf{BPRG} = (\mathsf{setup}, \mathsf{init}, \mathsf{next}, \mathsf{rec}_{\mathscr{B}})$ *is said to be* $(t, q, \delta, (\cdot, \epsilon))$*-secure if for all adversaries* $\mathscr{B}$ *running in time* t*, it holds that* $\mathsf{Adv}^{\mathsf{BPRG}.(\cdot)}_{\mathscr{B},q} \geq \epsilon$ *and* $(\mathsf{setup}, \mathsf{init}, \mathsf{next})$ *is a* (t, q, δ)*-secure* PRG.

Definition 13 ($(t, q, \delta, (\cdot, \epsilon))$-FWD-secure BPRG). *A* $\mathsf{BPRG} = (\mathsf{setup}, \mathsf{init}, \mathsf{next}, \mathsf{rec}_{\mathscr{B}})$ *is said to be* $(t, q, \delta, (\cdot, \epsilon))$-FWD*-secure if for all adversaries* $\mathscr{B}$ *running in time* t*, it holds that* $\mathsf{Adv}^{\mathsf{BPRG}.(\cdot)}_{\mathscr{B},q} \geq \epsilon$ *and* $(\mathsf{setup}, \mathsf{init}, \mathsf{next})$ *is a* (t, q, δ)-FWD*-secure* PRG.

2.5 Trapdoor Projection Maps for Bilinear Groups

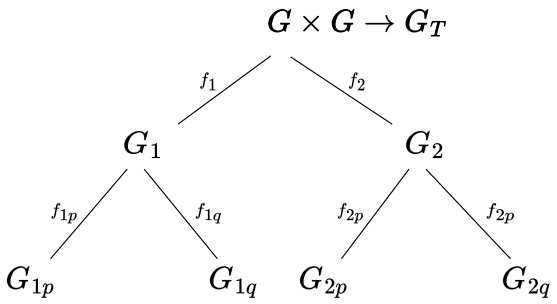

Fig. 1: TBG construction

Pairing-based cryptography is a striking illustration of the value of algebraic structure for constructing cryptographic schemes: a richer structure allows for a wider variety of cryptographic schemes, provided that there exists some hard problems on which security can be based. Groups with computable pairings have proved to be fruitful for designing cryptographic primitives and protocols. Meiklejohn et al. [24] give a surprising and unprecedented new structure of the pairing-friendly elliptic curve proposed by Boneh, Rubin, and Silverberg [4]. In

a nutshell, their new structure (later called TBG) can project a point from the group G onto its subgroup G_1 or G_2 with knowledge of a trapdoor.

We describe the TBG construction in Fig. 1. For a bilinear map $e : G \times G \to G_T$ (used especially for Weil pairing), TBG first generates parameters using the following setup algorithm.

Definition 14. $\mathsf{TBG.setup}(1^\lambda)$. *Use the algorithms in [24, Algorithm 1] to generate a prime q, a composite N, an elliptic curve E defined over $\mathbb{F}_q$ such that $G := E[N]$ contains N^2 points, and two distortion-free subgroups G_1 and G_2 of the N-torsion group G with their generators P_1 and P_2. Finally, let e be the Weil pairing and $G_T := \mu_N$. Output bilinear group public parameters $bgpp = (N, G_1, G_2, G_T, e, P_1, P_2)$.*

Because we do not need to discuss the underlying mathematics, we switch from additive to multiplicative notation so that we use g_1 and g_2 to represent the generators of G_1 and G_2, respectively. Therefore, $bgpp = (N, G_1, G_2, G_T, e, P_1, P_2)$ can be represented as $bgpp = (N, G_1, G_2, G_T, e, g_1, g_2)$.

Definition 15 (Trapdoor Projection Map [24]). *We say that a function $f : G \to G'$ is a projection map if it satisfies: (1) efficiently computable; (2) idempotent; and (3) $G' \subset G$. If, furthermore, f is assumed to be hard to compute without the knowledge of some additional piece of information, then we say that it is a trapdoor projection map.*

After generating $bgpp$, we can use the method in [24, section 3.1] to get a trapdoor projection map $\mathsf{td}_{\mathscr{S}} = f$. Then, we can map the elements of G to its torsion subgroups G_1 ($\mathsf{td}_{\mathscr{S}} = f_1$) and G_2 ($\mathsf{td}_{\mathscr{S}} = f_2$), and even further map the elements of G_1 and G_2 to their subgroups G_{1p} ($\mathsf{td}_{\mathscr{S}} = f_{1p}$), G_{1q}($\mathsf{td}_{\mathscr{S}} = f_{1q}$), G_{2p}($\mathsf{td}_{\mathscr{S}} = f_{2p}$), and G_{2q}($\mathsf{td}_{\mathscr{S}} = f_{2q}$). we have the following lemma 1 to guarantee the quality of the above mappings.

Lemma 1. *For $bgpp = (N, G_1, G_2, G_T, e, g_1, g_2) \leftarrow \mathsf{TBG.setup}$, $G_1 \cap G_2 = \mathcal{O}$, where $\mathcal{O} = End(E)$ is an order in some imaginary quadratic field K [24, Lemma 3.2.].*

The following properties of the $bgpp$ with trapdoor $\mathsf{td}_{\mathscr{S}} = f$ are used in this paper: (1) $u = f_1(u) \cdot f_2(u)$, for any $u \in G$; (2) for any $a, b \in G_1$ or $a, b \in G_2$, $e(a, b) = 1$; (3) for $u, v \in G$, $e(f_1(u), v) = e(u, f_2(v))$. Further, we use its special case $e(f_1(u), g) = e(u, g_2)$.

Note that $bgpp$ implies the group G and its generator g. When we obtain $bgpp$, we actually learn about G and g at the same time.

3 Asymmetric Trapdoor Pseudorandom Generators

3.1 Definitions

We first introduce the definitions of our ATPRG, then give its security definitions.

Definition 16. *An* ATPRG *has five algorithms* $(\mathsf{setup}, \mathsf{init}, \mathsf{PRGen}, \mathsf{SRGen}, \mathsf{rec}_{\mathscr{B}})$*, to simplify the description, we have omitted the random coins:*

- $(pp, \mathsf{td}_{\mathscr{B}}, \mathsf{td}_{\mathscr{S}}) \leftarrow \mathsf{setup}$*. Output a public parameter* pp*, a public trapdoor* $\mathsf{td}_{\mathscr{B}}$ *for recovering public pseudorandom sequence* $\overline{pr}$*, and a secret trapdoor* $\mathsf{td}_{\mathscr{S}}$ *for generating secret pseudorandom sequence* $\overline{sr}$*.*
- $s_0 \leftarrow \mathsf{init}(pp)$*. Output an initial state of* ATPRG*.*
- $(pr, s') \leftarrow \mathsf{PRGen}(pp, s)$*. Input a public parameter* pp *and a state* s*. Output a public pseudorandom number* pr *and the next state* s'*.*
- $sr \leftarrow \mathsf{SRGen}(pp, pr, \mathsf{td}_{\mathscr{S}})$*. Input a public parameter* pp*, a public pseudorandom number* pr*, and a secret trapdoor* $\mathsf{td}_{\mathscr{S}}$*. Output a secret pseudorandom number* sr*.*
- $\overline{pr} \leftarrow \mathsf{rec}_{\mathscr{B}}(pp, \mathsf{td}_{\mathscr{B}}, pr_n, n)$*. Input a public parameter* pp*, a public trapdoor* $\mathsf{td}_{\mathscr{B}}$*, a public pseudorandom number* pr_n *with its index* n*. Output a recovered sequence* $\overline{pr} = \{pr_1, \ldots, pr_n\}$ *of the public pseudorandom numbers. Note that the length of the sequence* $\overline{pr}$ *can be longer than* n*.*

In $\mathsf{rec}_{\mathscr{B}}$, the recovered sequence $\overline{pr}$ implies a sequence of the state values s_i, but the main role of our ATPRG is to recover the sequence of public pseudorandom numbers, so other recoverable information is ignored. ATPRG also exists two types of attackers: $\mathscr{A}$ and $\mathscr{B}$, and their behaviors are the same as BPRG. We expect ATPRG to be equivalent to a standard PRG for attacker $\mathscr{A}$, and ATPRG to be equivalent to a standard BPRG for attacker $\mathscr{B}$. Besides, we also expect that $\mathscr{A}$ and $\mathscr{B}$ cannot recover the sequence $\overline{sr}$ of the secret pseudorandom numbers by $\overline{pr}$ without the secret trapdoor $\mathsf{td}_{\mathscr{S}}$. We, therefore, have the following security definitions:

Definition 17 (ATPRG Inverse Advantage). *The inverse advantage of* $\mathscr{B}$ *against* ATPRG *is defined as follows:*

$$\mathsf{Adv}^{\mathsf{ATPRG.inv}}_{\mathscr{B},q} := \mathsf{Pr}\left[\begin{array}{c}(pp, \mathsf{td}_{\mathscr{B}}, \mathsf{td}_{\mathscr{S}}) \leftarrow \mathsf{setup},\\ s_0 \leftarrow \mathsf{init}(pp),\\ pr_1, \ldots, pr_q \leftarrow \mathsf{out}^q(\mathsf{PRGen}),\\ sr_1, \ldots, sr_q \leftarrow \mathsf{out}^q(\mathsf{SRGen}),\\ sr^*_1, \ldots, sr^*_q \leftarrow \mathscr{B}(pp, pr_1, \ldots, pr_q) :\\ sr_1, \ldots, sr_q = sr^*_1, \ldots, sr^*_q\end{array}\right].$$

Definition 18 (ATPRG Correctness Advantage). *The correctness advantage of* $\mathscr{B}$ *against* ATPRG *is defined as follows:*

$$\mathsf{Adv}^{\mathsf{ATPRG.crt}}_{\mathscr{B},q} := \mathsf{Pr}\left[\begin{array}{c}(pp, \mathsf{td}_{\mathscr{B}}, \mathsf{td}_{\mathscr{S}}) \leftarrow \mathsf{setup},\\ s_0 \leftarrow \mathsf{init}(pp),\\ pr_1, \ldots, pr_q \leftarrow \mathsf{out}^q(\mathsf{PRGen}),\\ sr_1, \ldots, sr_q \leftarrow \mathsf{out}^q(\mathsf{SRGen}),\\ sr^*_1, \ldots, sr^*_q \leftarrow \mathscr{B}(pp, \mathsf{td}_{\mathscr{S}}, pr_1, \ldots, pr_q) :\\ sr_1, \ldots, sr_q = sr^*_1, \ldots, sr^*_q\end{array}\right].$$

Note that the public pseudorandom sequence can be recovered completely (for $\overline{pr} = \{pr_1, \ldots, pr_n\}$, we can use public trapdoor and pr_i, $i \in [n]$, to recover a sequence $\overline{pr}^* = \{pr_1^*, \ldots, pr_n^*\}$ such that $\overline{pr} = \overline{pr}^*$) if $\mathsf{Adv}_{\mathscr{B},q}^{\mathsf{BPRG.first}} = 1$ or $\mathsf{Adv}_{\mathscr{B},q}^{\mathsf{BPRG.out}} = 1$. Therefore, we only define the correctness advantage of secret pseudorandom sequence.

An ATPRG should satisfy three properties: PR correctness, SR correctness, and non-reversibility. PR correctness requires that the public pseudorandom sequence can be recovered completely; specifically, it requires that $(\mathsf{setup}, \mathsf{init}, \mathsf{PRGen}, \mathsf{rec}_{\mathscr{B}})$ is a BPRG with $\mathsf{Adv}_{\mathscr{B},q}^{\mathsf{BPRG.first}} = 1$ or $\mathsf{Adv}_{\mathscr{B},q}^{\mathsf{BPRG.out}} = 1$. SR correctness requires that any user holding the secret trapdoor $\mathsf{td}_{\mathscr{S}}$ will compute the same sr_i for the same pr_i. Non-reversibility requires that $\psi : pr \to sr$ is computationally hard. If ψ is computationally easy, the adversary can easily use the public key to compute the secret key and make the cryptographic scheme based on ATPRG insecure. When an ATPRG satisfies all the above three properties, it has all the functions envisaged in Section 1. The formal definitions of these three properties are as follows:

Definition 19 (PR Correctness). *An* ATPRG *satisfies PR correctness if* $(\mathsf{setup}, \mathsf{init}, \mathsf{PRGen}, \mathsf{rec}_{\mathscr{B}})$ *is a* $(t, q, \delta, (first/out, 1))$*-secure* n*-outputs* BPRG *or* $(t, q, \delta, (first/out, 1))$-FWD*-secure* n*-outputs* BPRG.

Definition 20 (SR Correctness). *An* ATPRG *satisfies SR correctness if* $\mathsf{Adv}_{\mathscr{B},q}^{\mathsf{ATPRG.crt}}$ *is non-negligible.*

Definition 21 (Non-Reversibility). *An* ATPRG *satisfies non-reversibility if* $\mathsf{Adv}_{\mathscr{B},q}^{\mathsf{ATPRG.inv}}$ *is negligible.*

3.2 Construction

We use a (t, q, ϵ)-FWD-secure PRG $(\mathsf{setup}, \mathsf{init}, \mathsf{next})$, a (t, q, δ, ν)-IND$-CPA-secure reverse re-randomizable encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand}, \mathsf{Rand}^{-1})$, and TBG to construct our $\mathsf{ATPRG} = (\mathsf{setup}, \mathsf{init}, \mathsf{PRGen}, \mathsf{SRGen}, \mathsf{rec}_{\mathscr{B}})$. The details are shown in Fig. 2.

Theorem 1. *Our* ATPRG *instance satisfies PR correctness.*

Proof. From [11], the algorithms $(\mathsf{setup}, \mathsf{init}, \mathsf{PRGen}, \mathsf{rec}_{\mathscr{B}})$ of our ATPRG instance form a $(t, q, \delta, (first, 1))$-FWD-secure BPRG.

Theorem 2. *Our* ATPRG *instance satisfies SR correctness.*

Proof. $\mathsf{Adv}_{\mathscr{B},q}^{\mathsf{ATPRG.crt}} = 1$ since f_1 map is trapdoor projection maps.

Assumption 1 *Given* $bgpp \leftarrow \mathsf{TBG.setup}$ *and random* $u \xleftarrow{\$} G$ *for* $G : G_1 \times G_2$*, it is hard to compute* $f_1(u)$ *[24, Assumption 4.4].*

Theorem 3. *Our* ATPRG *instance satisfies non-reversibility.*

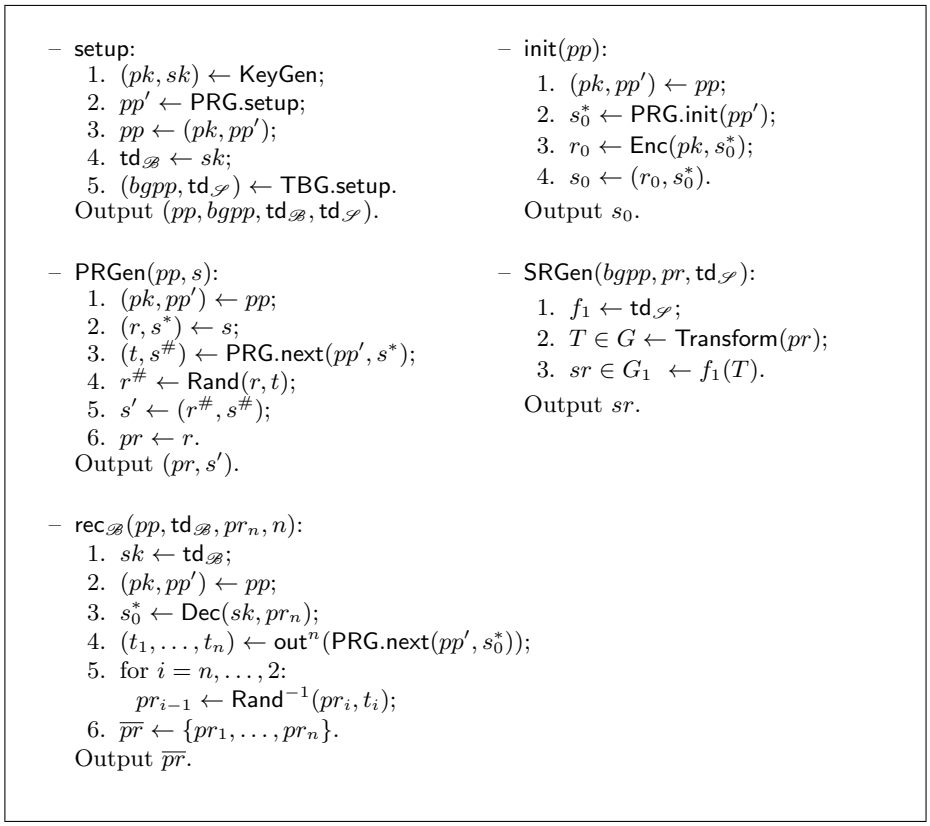

Fig. 2: ATPRG construction

Proof. From assumption 1, $\mathsf{Adv}^{\mathsf{ATPRG.inv}}_{\mathscr{B},q} = \eta(\lambda)$.

The type of pr values depends on the type of the ciphertexts of the reverse re-randomizable encryption scheme. The type of sr values is not only can be the torsion subgroup G_1 but also can be other torsion subgroups $G_2, G_{1p}, G_{2p}, G_{1q}, G_{2q}$ (mapping by the secret trapdoors $f_2, f_{1p}, f_{2p}, f_{1q}, f_{2q}$). For different (pairing-based) cryptographic schemes, we can use pr as public key, then use a hash function $H : \{0,1\}^{|pr|} \rightarrow G$ as the $\mathsf{Transform}$ method to transform pr to the elements of G, and finally, use the secret trapdoor to get sr as secret key. The hash function does not affect the compressibility of the sequence of pr values, and the advantage of $\mathsf{Adv}^{\mathsf{ATPRG.inv}}_{\mathscr{B},q}$. Further, we can use $(pp, \mathsf{td}_{\mathscr{B}}, pr_n, n)$ as public keys and $(bgpp, \mathsf{td}_{\mathscr{S}})$ as secret keys to reduce the pk and sk sizes by adding some bearable computational overheads. In addition, we can also directly use the reverse re-randomizable encryption scheme whose ciphertext belongs to group G to cancel hash functions. However, this approach requires a restriction on the types of encryption schemes. In contrast, using hash functions as the $\mathsf{Transform}$

method, an ATPRG can be constructed using any (t, q, δ, ν)-IND$-CPA-secure reverse re-randomizable encryption scheme.

4 Homomorphic Signature Scheme with Shorter Public Keys

In this section, we first introduce the definitions of HS scheme. Then, we give the construction of our HS scheme with $O(T)$-sized public keys. Specifically, in Section 4.2, we use the complete ATPRG algorithm to construct the HS scheme; in Section 4.3, we use some unique properties of homomorphic signature to simplify the scheme and eliminate the requirement of the public trapdoor. Finally, we prove the unforgeability of our scheme based on the DL and CDH assumptions in the standard model.

4.1 Definitions

We first give the definitions of labeled programs, multi-labeled programs [16], and well-defined programs [6].

Definition 22 (Labeled Programs). *A labeled program $\mathcal{P}$ includes a tuple $(f, \tau_1, \ldots, \tau_n)$, where $f : \mathcal{L}^n \to \mathcal{L}$ and τ_i is the label of the input of f. For labeled programs $\mathcal{P}_1, \ldots, \mathcal{P}_t$, they have a composed program $\mathcal{P}^*$ of a function $g : \mathcal{L}^t \to \mathcal{L}$ such that $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$.*

Definition 23 (Multi-labeled Programs). *A multi-labeled program $\mathcal{P}_\Delta$ includes a tuple $(\mathcal{P}, \Delta)$, where $\mathcal{P}$ is a labeled program $(f, \tau_1, \ldots, \tau_n)$ and Δ is a dataset identifier. For multi-labeled programs $(\mathcal{P}_1, \Delta), \ldots, (\mathcal{P}_t, \Delta)$, they have a composed program $\mathcal{P}^*$ of a function $g : \mathcal{L}^t \to \mathcal{L}$ within the same dataset such that $\mathcal{P}^*_\Delta = (\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t), \Delta)$.*

Definition 24 (Well-defined Programs). *Let $T_\Delta = \{(\tau_1, m_1), \ldots, (\tau_n, m_n)\}$ represents a list of the dataset Δ. A labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ is well-defined with respect to T_Δ if either one of the following cases holds:*

1. *For the existing messages $m_1, \ldots, m_n$, T_Δ contains all tuples $(\tau_1, m_1), \ldots, (\tau_n, m_n)$, which means that the entire input space of f for the dataset Δ has been authenticated.*
2. *For the tuple $(\tau_j, m_j) \notin T_\Delta$, the output space of the function $f(\{m_i\}_{(\tau_i, m_i) \in T_\Delta} \cup \{m_j\}_{(\tau_j, m_j) \notin T_\Delta})$ is equal to the output space of the function $f(\{m_i\}_{(\tau_i, m_i) \in T_\Delta})$.*

Definition 25 (Homomorphic Signature for Multi-labeled Programs). *A* HS *scheme has four PPT algorithms* (KeyGen, Sign, Eval, Ver)*:*

- $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$. *Input a security parameter λ. Output a secret key sk, a public key pk.*
- $\sigma \leftarrow \mathsf{Sign}(sk, L, m)$. *Input the secret key sk, a message m with its multi-label $L = (\Delta, \tau)$. Output a signature σ.*

- $\sigma \leftarrow \mathsf{Eval}(pk, f, \vec{\sigma})$. *Input the public key* pk*, a computed function* f*, and a set of signatures* $\vec{\sigma} = \sigma_1, \ldots, \sigma_n$ *corresponding to the inputs of* f *(assuming that* f *takes* n *inputs), where* σ_i *is generated by the* Sign *algorithm for any* $i \in [n]$*. Output an evaluation signature* σ*.*
- $0/1 \leftarrow \mathsf{Ver}(pk, \mathcal{P}_\Delta, \overline{m}, \sigma)$. *Input the public key* pk*, a computed result* $\overline{m}$ *with its evaluation signature* σ*, and the multi-labeled program* $\mathcal{P}_\Delta$ *corresponding to* σ*. Output* 1 *represents that* $\overline{m}$ *is the correct result of* $f(m_1, \ldots, m_n)$ *and* 0 *is incorrect.*

HS schemes have four properties: signature correctness, evaluation correctness, succinctness, and security (unforgeability).

Definition 26 (Signature Correctness). *A* HS *scheme satisfies signature correctness if for a given label space* $\mathcal{L}$*, all key pairs* $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$*, any label* $\tau \in \mathcal{L}$*, data identifier* $\Delta \in \{0,1\}^*$*, and any signature* $\sigma \leftarrow \mathsf{Sign}(sk, L_i, m_i)$*,* $1 \leftarrow \mathsf{Ver}(pk, \mathcal{P}_\Delta, m_i, \sigma_i)$ *holds with all but negligible probability.*

Definition 27 (Evaluation Correctness). *A* HS *scheme satisfies evaluation correctness if for a key pair* $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$*, any set of* $\{\mathcal{P}_i, m_i, \sigma_i\}_{i=1}^t$ *such that* $1 \leftarrow \mathsf{Ver}(pk, \mathcal{P}, m, \sigma)$*, and a function* $g : \mathcal{M}^t \to \mathcal{M}$*,* $1 \leftarrow \mathsf{Ver}(pk, \mathcal{P}^*, m^*, \sigma^*)$ *holds with all but negligible probability, where* $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$*,* $m^* = g(m_1, \ldots, m_t)$*, and* $\sigma^* = \mathsf{Eval}(pk, g, \sigma_1, \ldots, \sigma_t)$*.*

Definition 28 (Succinctness). *A* HS *scheme satisfies succinctness if for a fixed security parameter* λ*, the size of the signatures depends at most logarithmically on the dataset size.*

To define the security of the HS scheme, we need to define an experiment $\mathsf{Exp}^{\mathsf{EUF-CMA}}_{\mathscr{A},\mathsf{HS}}$ first. Note that, compared with $\mathsf{Exp}^{\mathsf{EUF-CMA}}_{\mathscr{A},\mathsf{HA}}$, this experiment does not need Ver queries since the HS scheme is public verified.

Definition 29. *The experiment* $\mathsf{Exp}^{\mathsf{EUF-CMA}}_{\mathscr{A},\mathsf{HS}}$ *includes three steps* (Setup, Sign queries, Forgery)*:*

- Setup*: The challenger runs* $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and gives* pk *to the adversary* $\mathscr{A}$*.*
- Sign queries*: The adversary* $\mathscr{A}$ *asks for signatures on* (L, m)*, where* $L = (\Delta, \tau)$*. If* (L, m) *is the first query with the dataset* Δ*, then the challenger initializes an empty list* $T_\Delta = \emptyset$*. If* $(\tau, m) \notin T_\Delta$*, the challenger runs* $\sigma \leftarrow \mathsf{Sign}(sk, L, m)$*, gives* σ *to* $\mathscr{A}$*, and adds* (τ, m) *to* T_Δ*. If* $(\tau, m) \in T_\Delta$*, the challenger gives the corresponding signature* σ *to* $\mathscr{A}$*. If, for* (τ, m)*,* $\tau \in T_\Delta$ *but* $m \neq m^*$ *(*m^* *is the corresponding message of* τ *in* T_Δ*), the challenger does not reply.*
- Forgery*: The adversary* $\mathscr{A}$ *finally returns a forgery tuple* $(\mathcal{P}^*_\Delta, \overline{m}^*, \sigma^*)$*. The output of the experiment is 1 iff* $1 \leftarrow \mathsf{Ver}(pk, \mathcal{P}^*_\Delta, \overline{m}^*, \sigma^*)$ *and one of the following three forgery types holds.*
  1. *No list* T_Δ *was created during the experiment.*
  2. $\mathcal{P}^*$ *is well-defined and* $\overline{m}^* \neq f^*(\{m_i\}_{(\tau_i, m_i) \in T_{\Delta^*}})$*.*

3. $\mathcal{P}^*$ is not well-defined.

Definition 30 (Security (Unforgeability)). *A* HS *scheme satisfies unforgeability if the following equation holds.*

$$\mathsf{Pr}[\mathsf{Exp}^{\mathsf{EUF-CMA}}_{\mathscr{A},\mathsf{HS}} = 1] \leq \eta(\lambda). \tag{1}$$

Besides, some HS schemes may have an extra property: efficient verification, which improves verification efficiency (in the sense of amortization):

Definition 31 (Efficient Verification). *A* HS *scheme for multi-labeled programs satisfies efficient verification, if there exists two algorithms* (VerPerp, EffVer) *such that:*

- $vk_{\mathcal{P}} \leftarrow \mathsf{VerPerp}(pk, \mathcal{P})$. *Input the public key pk and the labeled program* $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$. *Output a concise verification key* $vk_{\mathcal{P}}$ *which does not depend on any dataset identifier* Δ.
- $0/1 \leftarrow \mathsf{EffVer}(pk, vk_{\mathcal{P}}, \Delta, \overline{m}, \sigma)$. *Input the public key pk, the concise verification key* $vk_{\mathcal{P}}$, *a dataset identifier* Δ, *and a computed result* $\overline{m}$ *with its evaluation signature* σ. *Output 0 (reject) or 1 (accept).*

VerPerp *and* EffVer *are required to satisfy the following properties:*

1. ***Correctness***. *Let* $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$ *be honestly generated keys, given any correct tuple* $(\mathcal{P}_\Delta, \overline{m}, \sigma)$ *such that* $1 \leftarrow \mathsf{Ver}(pk, \mathcal{P}_\Delta, \overline{m}, \sigma)$, *for every* $vk_{\mathcal{P}} \leftarrow \mathsf{VerPerp}(pk, \mathcal{P})$, *we have* $\mathsf{Pr}[\mathsf{EffVer}(pk, vk_{\mathcal{P}}, \Delta, \overline{m}, \sigma) = 1] = 1$.
2. ***Amortized Efficiency***. *Given* $\mathcal{P}_\Delta = (f, \tau_1, \ldots, \tau_n, \Delta)$ *and valid input messages* $m_1, \ldots, m_n$, *let* $t(n)$ *be the time of compute* $\mathcal{P}(m_1, \ldots, m_n)$, *we require that the time of* $\mathsf{EffVer}(pk, vk_{\mathcal{P}}, \Delta, \overline{m}, \sigma)$ *is independent of* n, *i.e.,* $O(1)$.

4.2 Construction

This section mainly shows a general approach constructing HS scheme using ATPRG, which can provide design ideas for other cryptographic schemes based on ATPRG. Specifically, we generate secret pseudorandom sequence $\overline{sr} = \{S_1, \ldots, S_n\}$ by a public pseudorandom sequence $\overline{pr} = \{R_1, \ldots, R_n\}$, therefore, the public trapdoor $\mathsf{td}_{\mathscr{B}}$ and R_i for any $i \in [n]$ can be used to recover the whole sequence $R_1, \ldots, R_n$. We use a hash function $H : \{0,1\}^* \to G$ to construct ATPRG.SRGen algorithm, the details of which are as follows:

- $\mathsf{SRGen}(bgpp, R_i, \mathsf{td}_{\mathscr{S}})$:
  1. $f_1 \leftarrow \mathsf{td}_{\mathscr{S}}$;
  2. $T \in G \leftarrow H(R_i)$;
  3. $S_i \in G_1 \leftarrow f_1(T)$.

  Output $S_i = f_1(H(R_i))$.

We construct the HS scheme upon the work of Catalano et al. [7], and the regular digital signature scheme Σ' used in our scheme is the same as theirs. Thus, following [7], the public key length of Σ' can be regarded as $O(1)$ size when computing the public key size. Roughly speaking, the main difference between our scheme and [7] is: Catalano et al. introduce the notion of asymmetric programmable hash functions (APHFs) and design a HS scheme with short public key size based on APHFs, which reduced the public key size of HS from $O(N+T)$ to $O(\sqrt{N}+\sqrt{T})$; we introduce a new primitive ATPRG, and then replace APHFs in their scheme with ATPRG to obtain a HS scheme with shorter public key size, which further reduces the public key size of HS from $O(\sqrt{N}+\sqrt{T})$ to $O(T)$. Note that N is dataset size and T is data dimension; intuitively, many practical large-scale datasets should have N much larger than T.

Our homomorphic signature scheme $\mathsf{HS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Ver})$ is as follows:

- $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$:
  1. $(pp, bgpp, \mathsf{td}_{\mathscr{B}}, \mathsf{td}_{\mathscr{S}}) \leftarrow \mathsf{ATPRG.setup}$, where $bgpp = (N, G_1, G_2, G_T, e, g_1, g_2)$, then run $\mathsf{ATPRG.init}$;
  2. Define a hash function $H : \{0,1\}^* \rightarrow G$ and a pseudorandom function $F_K : \{0,1\}^* \rightarrow Z_N$, then choose a random seed $K \xleftarrow{\$} \mathcal{K}$ for F_K;
  3. Randomly choose T elements $h_1, \ldots, h_T$ from G_1 group, where T is the dimension of message m, let $h = (h_1, \ldots, h_T)$;
  4. Give a regular digital signature scheme $\Sigma' = (\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Ver}')$, and generate a key pair $(pk', sk') \leftarrow \mathsf{KeyGen}'(1^\lambda)$ for the regular signature scheme;
  5. Run $\mathsf{ATPRG.PRGen}$ to get $R_1, \ldots, R_n$ such that we have $R_1, \ldots, R_n \leftarrow \mathsf{rec}_{\mathscr{B}}(pp, \mathsf{td}_{\mathscr{B}}, R_n, n)$, where $R_i \in \{0,1\}^*$ for $i \in [n]$;
  6. $sk \leftarrow (bgpp, pp, K, \mathsf{td}_{\mathscr{S}}, sk')$, $pk \leftarrow (bgpp, pp, \mathsf{td}_{\mathscr{B}}, pk', h, R_n)$.
- $\sigma \leftarrow \mathsf{Sign}(sk, L, m_i)$: To sign a message $m_i \in \mathbb{Z}_N$ with its multi-label $L = (\Delta, \tau_i)$, proceed as follows.
  1. $\alpha \leftarrow \mathsf{F}_K(\Delta)$, compute $A^{(i)} = (A_1^{(i)}, A_2^{(i)}) = (g^\alpha, g_2^\alpha)$ and $\sigma_\Delta^{(i)} \leftarrow \mathsf{Sign}'(sk', \Delta | A)$;
  2. Run $f_1(H(R_i)) \in G_1 \leftarrow \mathsf{ATPRG.SRGen}(bgpp, R_i, \mathsf{td}_{\mathscr{S}})$, select $W_i \xleftarrow{\$} G_1$, then compute
$$U_i = \left(f_1(H(R_i)) \cdot \prod_{j=1}^{T} h_j^{m_i[j]} \cdot W_i \right)^\alpha;$$
  3. $\sigma_i \leftarrow (\sigma_\Delta^{(i)}, A^{(i)}, U_i, W_i)$.
- $\sigma \leftarrow \mathsf{Eval}(pk, f, \vec{\sigma})$: for a linear function $f : \mathbb{Z}_N^n \rightarrow \mathbb{Z}_N$ described by its coefficients $f = (c_1, \ldots, c_n)$
  1. $(\sigma_1, \ldots, \sigma_n) \leftarrow \vec{\sigma}$ where $\sigma_i = (\sigma_\Delta^{(i)}, A^{(i)}, U_i, W_i)$ for all $i \in [n]$;
  2. Compute
$$\overline{U} = \prod_{i=1}^{n} U_i^{c_i}, \ \overline{W} = \prod_{i=1}^{n} W_i^{c_i};$$

3. $\sigma \leftarrow (\sigma_{\Delta} = \sigma_{\Delta}^{(1)}, A = A^{(1)}, \overline{U}, \overline{W})$.

– $0/1 \leftarrow \mathsf{Ver}(pk, \mathcal{P}_{\Delta}, \overline{m}, \sigma)$: for verifying the correctness of the computed result $\overline{m} = \sum_{i=1}^{n} c_i \cdot m_i$,
  1. Run $\mathsf{rec}_{\mathscr{B}}(pp, \mathsf{td}_{\mathscr{B}}, R_n, n)$ to recover $R_1, \ldots, R_n$;
  2. Run $\mathsf{Ver}'(pk', \Delta | A, \sigma_{\Delta})$ to check whether σ_{Δ} is valid, if not, stop and output 0. Otherwise, output 1 iff the following equations are satisfied:

$$e(\overline{U}, g_1) = 1, \tag{2}$$

$$e\left(\prod_{i=1}^{n} H(R_i)^{c_i}, A_2\right) \cdot e\left(\prod_{j=1}^{T} h_j^{\overline{m}[j]}, A_1\right) \cdot e\left(\prod_{i=1}^{n} W_i^{c_i}, A_1\right) = e(\overline{U}, g). \tag{3}$$

Comparison of public key size: The public key of [7] is $(pk', bgpp, pek, pek')$, where pk' is the public key of the regularly digital signature scheme, $bgpp = (p, G_1, G_2, G_T, g_1, g_2, e)$ is the parameters of bilinear group, $pek = \{A_i, B_i\}_{i=1}^{\lceil \sqrt{N} \rceil}$, and $pek' = \{A'_j, B'_j\}_{j=1}^{\lceil \sqrt{T} \rceil}$. Therefore, [7] has $O(\sqrt{N} + \sqrt{T})$-sized public keys. The public key of our construction is $pk = (bgpp, pp, \mathsf{td}_{\mathscr{B}}, pk', h, R_n)$, where the size of $(bgpp, pp, pk', \mathsf{td}_{\mathscr{B}}, R_n)$ is $O(1)$, and $h = (h_1, \ldots, h_T)$. Therefore, our HS scheme only has $O(T)$-sized public keys, independent of the dataset size.

Comparison of secret key size: The secret key of [7] is $(sk', K, \hat{K}, sek, sek')$, where sk' is the secret key of the regularly digital signature scheme, K and $\hat{K}$ are the two random seeds, $sek = \{\alpha_i, \beta_i\}_{i=1}^{\lceil \sqrt{N} \rceil}$, and $sek' = \{\alpha'_j, \beta'_j\}_{j=1}^{\lceil \sqrt{T} \rceil}$. Therefore, [7] also has $O(\sqrt{N} + \sqrt{T})$-sized secret keys. The secret key of our HS scheme is $sk = (bgpp, pp, K, \mathsf{td}_{\mathscr{S}}, sk')$, which is $O(1)$ size. Hence, Our HS scheme has more advantages of applying to the single user with large-scale datasets.

Theorem 4. *The scheme satisfies efficient verification.*

Proof. The verification algorithm can be rewritten as:

– $vk_{\mathcal{P}} \leftarrow \mathsf{VerPerp}(pk, \mathcal{P})$. Input the public key pk and $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$. Run $\mathsf{rec}_{\mathscr{B}}(pp, \mathsf{td}_{\mathscr{B}}, R_n, n)$ to recover $R_1, \ldots, R_n$, compute $vk_{\mathcal{P}} = f(H(R_1), \ldots, H(R_n)) = \prod_{i=1}^{n} H(R_i)^{c_i}$. Output a concise verification key $vk_{\mathcal{P}}$.
– $0/1 \leftarrow \mathsf{EffVer}(pk, vk_{\mathcal{P}}, \Delta, \overline{m}, \sigma)$. Run $\mathsf{Ver}'(pk', \Delta | A, \sigma_{\Delta})$ to check whether σ_{Δ} is valid, if not, stop and output 0. Otherwise, output 1 iff the following equations are satisfied:

$$e(\overline{U}, g_1) = 1, \tag{4}$$

$$e\left(vk_{\mathcal{P}}, A_2\right) \cdot e\left(\prod_{j=1}^{T} h_j^{\overline{m}[j]}, A_1\right) \cdot e\left(\prod_{i=1}^{n} W_i^{c_i}, A_1\right) = e(\overline{U}, g). \tag{5}$$

Theorem 5. *The scheme satisfies succinctness.*

Proof. The signature size of our HS scheme is independent of the dataset size.

Theorem 6. *The scheme satisfies signature correctness.*

Proof. Let $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$ be an honestly generated key pair with $(sk = (bgpp, pp, K, \mathsf{td}_{\mathscr{S}}, sk'), pk = (bgpp, pp, \mathsf{td}_{\mathscr{B}}, pk', h, R_n))$ and let $\sigma_i \leftarrow \mathsf{Sign}(sk, L, m_i)$ be a honestly generated signature. For $\mathsf{Ver}(pk, \mathcal{P}_\Delta, m_i, \sigma)$, we have

$$e(U_i, g_1) = 1,$$

since $U_i \in G_1$. And

$$\begin{aligned}
&e(R_i, A_2) \cdot e\left(\prod_{j=1}^{T} h_j^{m_i[j]}, A_1\right) \cdot e(W_i, A_2) \\
&= e(R_i, g_2^\alpha) \cdot e\left(\prod_{j=1}^{T} h_j^{m_i[j]}, g^\alpha\right) \cdot e(W_i, g^\alpha) \\
&= e(f_1(R_i), g)^\alpha \cdot e\left(\prod_{j=1}^{T} h_j^{m_i[j]}, g\right)^\alpha \cdot e(W_i, g)^\alpha \\
&= e\left(\left(f_1(R_i) \cdot \prod_{j=1}^{T} h_j^{m_i[j]} \cdot W_i\right)^\alpha, g\right) \\
&= e(U_i, g).
\end{aligned}$$

Theorem 7. *The scheme satisfies evaluation correctness.*

Proof. Let $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$ be an honestly generated key pair with $(sk = (bgpp, pp, K, \mathsf{td}_{\mathscr{S}}, sk'), pk = (bgpp, pp, \mathsf{td}_{\mathscr{B}}, pk', h, R_n))$. Given $\{\mathcal{P}_{i,\Delta}, m_i, \sigma_i\}_{i=1}^t$ such that $1 \leftarrow \mathsf{Ver}(pk, \mathcal{P}_{i,\Delta}, m_i, \sigma_i)$, for all $i = 1$ to t. Let $\sigma \leftarrow \mathsf{Eval}(pk, f, \sigma_1, \ldots, \sigma_t)$. Finally, we want to prove that the verification algorithm $\mathsf{Ver}(pk, \mathcal{P}_\Delta, \overline{m}, \sigma)$ outputs 1.

Since each σ_i verifies correctly, we have $\overline{U} = \prod_{i=1}^{t} U_i^{c_i} \in G_1$, thus $e(\overline{U}, g_1) = 1$. And for equation 3, we have

$$
\begin{aligned}
& e\left(\prod_{i=1}^{t} R_i^{c_i}, A_2\right) \cdot e\left(\prod_{j=1}^{T} h_j^{\overline{m}[j]}, A_1\right) \cdot e\left(\prod_{i=1}^{t} W_i^{c_i}, A_1\right) \\
&= e\left(\prod_{i=1}^{t} R_i^{c_i}, g_2^{\alpha}\right) \cdot e\left(\prod_{j=1}^{T} h_j^{\overline{m}[j]}, g^{\alpha}\right) \cdot e\left(\prod_{i=1}^{t} W_i^{c_i}, g^{\alpha}\right) \\
&= e\left(\prod_{i=1}^{t} R_i^{c_i}, g_2\right)^{\alpha} \cdot e\left(\prod_{i=1}^{t}\left(\prod_{j=1}^{T} h_j^{m_i[j]}\right)^{c_i}, g\right)^{\alpha} \cdot e\left(\prod_{i=1}^{t} W_i^{c_i}, g\right)^{\alpha} \\
&= e\left(\prod_{i=1}^{t} f_1(R_i)^{c_i}, g\right)^{\alpha} \cdot e\left(\prod_{i=1}^{t}\left(\prod_{j=1}^{T} h_j^{m_i[j]}\right)^{c_i}, g\right)^{\alpha} \cdot e\left(\prod_{i=1}^{t} W_i^{c_i}, g\right)^{\alpha} \\
&= e\left(\prod_{i=1}^{t}\left(\left(f_1\left(R_i\right) \cdot \prod_{j=1}^{T} h_j^{m_i[j]} \cdot W_i\right)^{\alpha}\right)^{c_i}, g\right) \\
&= e\left(\prod_{i=1}^{t} U_i^{c_i}, g\right) \\
&= e(\overline{U}, g).
\end{aligned}
$$

Theorem 8. *The scheme satisfies unforgeability if DL and CDH problems are hard, F_K is a pseudorandom function, and the one-way function exists for adversary $\mathscr{B}$.*

Proof. We can only deal with forgery 1 and 2 since forgery 3 can be converted into forgery 2 [7]. Our idea is similar to [7,26]. To prove this theorem, we need to define a series of games with the adversary $\mathscr{B}$ and we will show that the adversary $\mathscr{B}$ wins (the game outputs 1) only with negligible probability. Following the notation of [7], we also write $G_i(\mathscr{B})$ to denote that game i outputs 1, and bad_i represents the flag values of game i. bad_i initially sets to false, if at the end of the game any of these flags is set to true, the game outputs 0. We use Bad_i represents the event that bad_i is set to true during the game.

- **Game 1**: This game is the security experiment $\mathsf{Exp}_{\mathscr{B},\mathsf{HS}}^{\mathsf{EUF-CMA}}$, where $\mathscr{B}$ only outputs forgery 1 and 2.
- **Game 2**: This game is defined as Game 1 apart from the fact that whenever $\mathscr{B}$ outputs a forgery tuple $(\mathcal{P}_{\Delta}^{*}, \overline{m}^{*}, \sigma^{*})$, where $\sigma^{*} = (\sigma_{\Delta}^{*}, A^{*}, \overline{U}^{*}, \overline{W}^{*})$ such that A^{*} was not generated by the challenger, then Game 2 sets $\mathsf{bad}_2 = \mathsf{true}$.
- **Game 3**: This game is defined as Game 2 except that the pseudorandom function F_K of our scheme is replaced with a true random function $F : \{0,1\}^{*} \to \mathbb{Z}_N$.

– **Game 4**: This game is defined as Game 3 except for an additional check. For a forgery tuple $(\mathcal{P}^*_\Delta, \overline{m}^*, \sigma^*)$, it computes $\overline{m} = \sum_{i=1}^{n} c_i \cdot m_i$, and checks whether $\prod_{j=1}^{T} h_j^{\overline{m}[j]} = \prod_{j=1}^{T} h_j^{\overline{m}^*[j]}$. If it does, sets $\mathsf{bad}_4 = \mathsf{true}$.

Games 1 and 2 are only different if Bad_2 occurs, which means that the adversary $\mathscr{B}$ can obtain an existential forgery for the regular digital signature scheme. (EUF-CMA-secure digital signatures can be constructed from one-way functions [21, 25]). Games 2 and 3 are computationally indistinguishable when F_K is a pseudorandom function. Games 3 and 4 have the following relationship: $|\mathsf{Pr}[G_3(\mathscr{B})] - \mathsf{Pr}[G_4(\mathscr{B})]| \leq \mathsf{Pr}[\mathsf{Bad}_4]$. In Lemma 2 we show that Bad_4 is negligible for adversary $\mathscr{B}$ under DL assumption. Afterward, in Lemma 3, we show how a challenger can use an adversary $\mathscr{B}$ winning Game 4 to break the CDH assumption.

Lemma 2. $\mathsf{Pr}[\mathsf{Bad}_4] \leq \eta(\lambda)$ *if the DL assumption holds in* G_1.

Proof. Given $(g_1, g_1^a) \in G_1$, we show how the challenger to break DL assumption in G_1. Our simulation is similar to [26].

– Setup: The challenger selects an index $i \in [T]$ and runs $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$ except for the generation of the h. It chooses $x_1, \ldots, x_T \xleftarrow{\$} \mathbb{Z}_N$, sets $h_j = g_1^{x_j}$ for $j \neq i$ and sets $h_i = (g_1^a)^{x_i}$. This is perfectly indistinguishable from a real execution of this game since x_j is random. Finally, the challenger gives pk to the adversary $\mathscr{B}$.
– Sign queries: The challenger answers all queries of the adversary $\mathscr{B}$ faithfully.
– Forgery: The adversary $\mathscr{B}$ finally returns a forgery tuple $(\mathcal{P}^*_\Delta, \overline{m}^*, \sigma^*)$, where $\prod_{j=1}^{T} h_j^{\overline{m}[j]} = \prod_{j=1}^{T} h_j^{\overline{m}^*[j]}$.

For the forgery tuple $(\mathcal{P}^*_\Delta, \overline{m}^*, \sigma^*)$, the challenger checks whether $\overline{m}[i] \neq \overline{m}^*[i]$. If not, restarts the simulation. Otherwise, we have

$$\prod_{j=1}^{T} h_j^{\overline{m}[j]} = \prod_{j=1, j\neq i}^{T} g_1^{x_j \cdot \overline{m}[j]} \cdot g_1^{a \cdot x_i \cdot \overline{m}[j]} = \prod_{j=1, j\neq i}^{T} g_1^{x_j \cdot \overline{m}^*[j]} \cdot g_1^{a \cdot x_i \cdot \overline{m}^*[j]} = \prod_{j=1}^{T} h_j^{\overline{m}^*[j]}$$

$$\Rightarrow a \cdot x_i \cdot \overline{m}[i] + \sum_{j=1, j\neq i}^{T} x_j \cdot \overline{m}[j] = a \cdot x_i \cdot \overline{m}^*[i] + \sum_{j=1, j\neq i}^{T} x_j \cdot \overline{m}^*[j].$$

The challenger can compute

$$a = \frac{1}{x_i \cdot (\overline{m}^*[i] - \overline{m}[i])} \sum_{j=1, j\neq i}^{T} x_j \cdot (\overline{m}[j] - \overline{m}^*[j]).$$

Lemma 3. *The challenger can use an adversary* $\mathscr{B}$ *that wins in Game 4 to break the CDH assumption.*

Proof. Given $(bgpp = (N, G_1, G_2, G_T, e, g_1, g_2), g_1^a, g_1^b, g_2^b, g^b)$, where $a, b \xleftarrow{\$} \mathbb{Z}_N$, we show how the challenger to break CDH assumption in G_1.

– Setup: Let Q be the number of datasets in which the adversary $\mathscr{B}$ makes signature queries. The challenger guesses the dataset $\hat{\Delta}$ in which the adversary $\mathscr{B}$ produces a forgery, then runs $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$ except for the generation of the h. It choose $x_1, \ldots, x_T \xleftarrow{\$} \mathbb{Z}_N$, sets $h_j = (g_1^a)^{x_j}$ for $j \in [T]$. This is perfectly indistinguishable from an honest setup since x_j is random. Finally, the challenger gives pk to the adversary $\mathscr{B}$.

– Sign queries: The adversary $\mathscr{B}$ asks for signatures on (L, m), where $L = (\Delta, \tau)$.

1. If the adversary $\mathscr{B}$ queries signatures for the dataset $\Delta \neq \hat{\Delta}$. The challenger runs $\alpha \leftarrow \mathsf{F}_K(\Delta)$, sets $A = (A_1, A_2) = (g^\alpha, g_2^\alpha)$, and $\sigma_\Delta \leftarrow \mathsf{Sign}'(sk', \Delta|A)$. It selects $r \xleftarrow{\$} \mathbb{Z}_N$, sets $W = g_1^r \cdot g_1^{a \cdot \sum_{j=1}^T -x_j \cdot m[j]}$, computes

$$U = \left(f_1(H(R)) \cdot \prod_{j=1}^{T} h_j^{m[j]} \cdot W \right)^\alpha = \left(f_1(H(R)) \cdot g_1^r \right)^\alpha .$$

Finally, the challenger returns $\sigma = (\sigma_\Delta, A, U, W)$ to $\mathscr{B}$. The signature correctness can be easily verified: $e(U, g_1) = 1$ and

$$\begin{aligned}
& e(H(R), A_2) \cdot e\left(\prod_{j=1}^{T} h_j^{m[j]}, A_1 \right) \cdot e(W, A_1) \\
&= e(H(R), g_2^\alpha) \cdot e\left(\prod_{j=1}^{T} ((g_1^a)^{x_j})^{m[j]}, g^\alpha \right) \cdot e\left(g_1^r \cdot g_1^{a \cdot \sum_{j=1}^T -x_j \cdot m[j]}, g^\alpha \right) \\
&= e(H(R), g_2)^\alpha \cdot e(g_1^r, g)^\alpha \\
&= e\left((f_1(H(R)) \cdot g_1^r)^\alpha, g \right) \\
&= e(U, g).
\end{aligned}$$

2. If the adversary $\mathscr{B}$ queries signatures for the dataset $\Delta = \hat{\Delta}$. The challenger selects $y \xleftarrow{\$} \mathbb{Z}_N$, then sets $A = (A_1, A_2) = ((g^b)^y, (g_2^b)^y)$ and $\sigma_\Delta \leftarrow \mathsf{Sign}'(sk', \Delta|A)$. It selects $r \xleftarrow{\$} \mathbb{Z}_N$, sets $W = f_1(H(R))^{-1} \cdot g_1^r \cdot g_1^{a \cdot \sum_{j=1}^T -x_j \cdot m[j]}$ ($f_1(H(R))^{-1}$ represents the inverse of $f_1(H(R))$), computes $U = (g_1^b)^{r \cdot y}$. Finally, the challenger returns $\sigma = (\sigma_\Delta, A, U, W)$ to

$\mathscr{B}$. The signature correctness can be verified: $e(U, g_1) = 1$ and

$$\begin{aligned}
&e(H(R), A_2) \cdot e\left(\prod_{j=1}^{T} h_j^{m[j]}, A_1\right) \cdot e(W, A_1) \\
&= e(H(R), g_2^{b\cdot y}) \cdot e\left(f_1(H(R))^{-1} \cdot g_1^r \cdot g_1^{a\cdot\sum_{j=1}^{T} -x_j\cdot m[j]}, g^{b\cdot y}\right) \\
&\cdot e\left(\prod_{j=1}^{T} ((g_1^a)^{x_j})^{m[j]}, g^{b\cdot y}\right) \\
&= e(f_1(H(R)), g)^{b\cdot y} \cdot e(f_1(H(R))^{-1} \cdot g_1^r, g)^{b\cdot y} \\
&= e\left((g_1^b)^{r\cdot y}, g\right) \\
&= e(U, g).
\end{aligned}$$

– Forgery: The adversary $\mathscr{B}$ finally returns a forgery tuple $(\mathcal{P}_{\Delta}^*, \overline{m}^*, \sigma^*)$, where $\prod_{j=1}^{T} h_j^{\overline{m}[j]} \neq \prod_{j=1}^{T} h_j^{\overline{m}^*[j]}$ and $\sum_{j=1}^{T} x_j \cdot \overline{m}[j] \neq \sum_{j=1}^{T} x_j \cdot \overline{m}^*[j]$ since $\mathsf{bad}_4 = \mathsf{true}$.

$\overline{U}$ and $\overline{U}^*$ are valid signatures for the function f, we have

$$\begin{aligned}
\overline{U} &= \left(\prod_{i=1}^{n} f_1(H(R_i))^{c_i} \cdot \overline{W}\right)^{b\cdot y} \cdot \left(\prod_{j=1}^{T} h_j^{\overline{m}[j]}\right)^{b\cdot y} \\
&= \left(\prod_{i=1}^{n} f_1(H(R_i))^{c_i}\right)^{b\cdot y} \cdot \left(\prod_{i=1}^{n} W_i^{c_i}\right)^{b\cdot y} \cdot \left(g_1^{b\cdot y\cdot a\cdot\sum_{j=1}^{T} x_j\cdot\overline{m}[j]}\right)
\end{aligned}$$

and

$$\begin{aligned}
\overline{U}^* &= \left(\prod_{i=1}^{n} f_1(H(R_i))^{c_i} \cdot \overline{W}\right)^{b\cdot y} \cdot \left(\prod_{j=1}^{T} h_j^{\overline{m}^*[j]}\right)^{b\cdot y} \\
&= \left(\prod_{i=1}^{n} f_1(H(R_i))^{c_i}\right)^{b\cdot y} \cdot \left(\prod_{i=1}^{n} W_i^{c_i}\right)^{b\cdot y} \cdot \left(g_1^{b\cdot y\cdot a\cdot\sum_{j=1}^{T} x_j\cdot\overline{m}^*[j]}\right).
\end{aligned}$$

If compute $\overline{U} \cdot (\overline{U}^*)^{-1}$, we can get

$$\begin{aligned}
\frac{\overline{U}}{\overline{U}^*} &= \frac{\left(\prod_{i=1}^{n} f_1(H(R_i))^{c_i}\right)^{b\cdot y} \cdot \left(\prod_{i=1}^{n} W_i^{c_i}\right)^{b\cdot y} \cdot \left(g_1^{b\cdot y\cdot a\cdot\sum_{j=1}^{T} x_j\cdot\overline{m}[j]}\right)}{\left(\prod_{i=1}^{n} f_1(H(R_i))^{c_i}\right)^{b\cdot y} \cdot \left(\prod_{i=1}^{n} W_i^{c_i}\right)^{b\cdot y} \cdot \left(g_1^{b\cdot y\cdot a\cdot\sum_{j=1}^{T} x_j\cdot\overline{m}^*[j]}\right)} \\
&= g_1^{a\cdot b\cdot(y\cdot\sum_{j=1}^{T} x_j\cdot(\overline{m}[j]-\overline{m}^*[j]))}.
\end{aligned}$$

The challenger, therefore, can get g_1^{ab} by computing

$$\left(\frac{\overline{U}}{\overline{U}^*}\right)^{\frac{1}{y\cdot\sum_{j=1}^{T} x_j\cdot(\overline{m}[j]-\overline{m}^*[j])}} = g_1^{\frac{a\cdot b\cdot(y\cdot\sum_{j=1}^{T} x_j\cdot(\overline{m}[j]-\overline{m}^*[j]))}{y\cdot\sum_{j=1}^{T} x_j\cdot(\overline{m}[j]-\overline{m}^*[j])}} = g_1^{ab}.$$

Lemma 3 requires that $f_1(H(R))^{-1}$ is computationally easy if we know $H(R)$ and f_1. The following Lemma 4 shows that $f_1(H(R))^{-1}$ can be computed efficiently by the challenger.

Lemma 4. *The challenger can easily compute* $f_1(H(R))^{-1}$ *if it knows* $H(R)$ *and* f_1.

Proof. In a nutshell, f_1 map preserves the group structure, for an element $u \in G$ and its inverse $u^{-1} \in G$, $f_1(u)^{-1} = f_1(u^{-1})$. Therefore, the challenger can first compute $H(R)^{-1}$ (e.g., use extended Euclidean algorithm, $O(log\ n)$ computation complexity), then compute $f_1(H(R)^{-1})$ to obtain $f_1(H(R))^{-1}$ since $f_1(H(R)^{-1}) = f_1(H(R))^{-1}$. Specifically, from [24], the group G has a fact that $G = G_1 \oplus G_2$; in other words, G has exponent N but contains N^2 points, it can be represented as the product of two cyclic groups (G_1 and G_2) of order N. Therefore, we can write any element $g^a \in G$ uniquely as $g^a = g_1^b g_2^c$, where $b, c \in \mathbb{Z}/N\mathbb{Z}$, $g_1 = g^{\frac{\sigma+s}{2s}}$, and $g_2 = g^{\frac{s-\sigma}{2s}}$ ($\frac{\sigma+s}{2s}$ and $\frac{s-\sigma}{2s}$ involve knowledge of the underlying elliptic curve, we will not explain their construction in detail. The readers can read [24] for the technical details, and treat them as constants in this proof). The computation of f_1 map can be described as $f_1(g^a) = (g^a)^{\frac{\sigma+s}{2s}} = g_1^a$. The challenger knows the values $H(R)$ and f_1, therefore, it also knows the inverse $H(R)^{-1}$. We write $H(R)$ as g^a and $H(R)^{-1}$ as g^{-a}, then their f_1 map are $f_1(g^a) = g_1^a$ and $f_1(g^{-a}) = g_1^{-a}$, respectively. Hence, the challenger can easily compute $f_1(H(R))^{-1}$ if it knows $H(R)$ and f_1.

4.3 Other Constructions

We discover that the public labels of HS have the same size as the dataset, and we can use this feature to simplify the scheme in the previous section. Note that this simplification does not apply to all cryptographic schemes and is only customized for HS. Specifically, we can also borrow labels $\tau_1, \ldots, \tau_n$ to generate secret pseudorandom sequence $\overline{sr}$. In other words, instead of running $\mathsf{ATPRG.PRGen}$ to generate public pseudorandom sequence $\overline{pr}$, we borrow the published labels as $\overline{pr}$. The modified construction is as follows:

- $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$:
  1. $(pp, bgpp, \mathsf{td}_{\mathscr{B}} = \perp, \mathsf{td}_{\mathscr{S}}) \leftarrow \mathsf{ATPRG.setup}$, where $bgpp = (N, G_1, G_2, G_T, e, g_1, g_2)$, then run $\mathsf{ATPRG.init}$;
  2. Define a hash function $H : \{0,1\}^* \rightarrow G$ and a pseudorandom function $F_K : \{0,1\}^* \rightarrow Z_N$, then choose a random seed $K \xleftarrow{\$} \mathcal{K}$ for F_K;
  3. Randomly choose $h = (h_1, \ldots, h_T)$ from G_1, where T is the dimension of message m;
  4. Give a regular digital signature scheme $\Sigma' = (\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Ver}')$, and generate a key pair $(pk', sk') \leftarrow \mathsf{KeyGen}'(1^\lambda)$ for the regular signature scheme;
  5. $sk \leftarrow (bgpp, pp, K, \mathsf{td}_{\mathscr{S}}, sk')$, $pk \leftarrow (bgpp, pp, \mathsf{td}_{\mathscr{B}}, pk', h)$.

– $\sigma \leftarrow \mathsf{Sign}(sk, L, m_i)$: To sign a message $m_i \in \mathbb{Z}_N$ with its multi-label $L = (\Delta, \tau_i)$, proceed as follows.
  1. $\alpha \leftarrow \mathsf{F}_K(\Delta)$, compute $A^{(i)} = (A_1^{(i)}, A_2^{(i)}) = (g^\alpha, g_2^\alpha)$ and $\sigma_\Delta^{(i)} \leftarrow \mathsf{Sign}'(sk', \Delta|A)$;
  2. Select $W_i \xleftarrow{\$} G_1$ and run $f_1(H(\tau_i)) \leftarrow \mathsf{ATPRG.SRGen}(bgpp, \tau_i, \mathsf{td}_{\mathscr{S}})$, then compute
$$U_i = \left(f_1(H(\tau_i)) \cdot \prod_{j=1}^{T} h_i^{m_i[j]} \cdot W_i \right)^\alpha;$$
  3. $\sigma_i \leftarrow (\sigma_\Delta^{(i)}, A^{(i)}, U_i, W_i)$.

– $\sigma \leftarrow \mathsf{Eval}(pk, f, \vec{\sigma})$: for a linear function $f : \mathbb{Z}_N^n \to \mathbb{Z}_N$ described by its coefficients $f = (c_1, \ldots, c_n)$
  1. $(\sigma_1, \ldots, \sigma_n) \leftarrow \vec{\sigma}$ where $\sigma_i = (\sigma_\Delta^{(i)}, A^{(i)}, U_i, W_i)$ for all $i \in [n]$;
  2. Compute
$$\overline{U} = \prod_{i=1}^{n} U_i^{c_i}, \ \overline{W} = \prod_{i=1}^{n} W_i^{c_i};$$
  3. $\sigma \leftarrow (\sigma_\Delta = \sigma_\Delta^{(1)}, A = A^{(1)}, \overline{U}, \overline{W})$.

– $0/1 \leftarrow \mathsf{Ver}(pk, \mathcal{P}_\Delta, \overline{m}, \sigma)$: for verifying the correctness of the computed result $\overline{m} = \sum_{i=1}^{n} c_i \cdot m_i$,
  1. Run $\mathsf{Ver}'(pk', \Delta|A, \sigma_\Delta)$ to check whether σ_Δ is valid, if not, stop and output 0. Otherwise, output 1 iff the following equations are satisfied:
$$e(\overline{U}, g_1) = 1, \tag{6}$$
$$e\left(\prod_{i=1}^{n} (H(\tau_i))^{c_i}, A_2 \right) \cdot e\left(\prod_{j=1}^{T} h_j^{\overline{m}[j]}, A_1 \right) \cdot e(\overline{W}, A_1) = e(\overline{U}, g). \tag{7}$$

Comparison of public key size: The public key of this construction is $pk = (bgpp, pp, \mathsf{td}_{\mathscr{B}}, pk', h)$, where the size of $(bgpp, pp, pk', \mathsf{td}_{\mathscr{B}})$ is $O(1)$, and $h = (h_1, \ldots, h_T)$. Therefore, this scheme also has $O(T)$-sized public keys, independent of the dataset size.

Theorem 9. *The scheme satisfies efficient verification.*

Proof. The verification algorithm can be rewritten as:

– $vk_{\mathcal{P}} \leftarrow \mathsf{VerPerp}(pk, \mathcal{P})$. Input the public key pk and $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$. Set $H(\tau_1), \ldots, H(\tau_n)$, compute $vk_{\mathcal{P}} = f(H(\tau_1), \ldots, H(\tau_n)) = \prod_{i=1}^{n} (H(\tau_i))^{c_i}$. Output a concise verification key $vk_{\mathcal{P}}$.
– $0/1 \leftarrow \mathsf{EffVer}(pk, vk_{\mathcal{P}}, \Delta, \overline{m}, \sigma)$. Run $\mathsf{Ver}'(pk', \Delta|A, \sigma_\Delta)$ to check whether σ_Δ is valid, if not, stop and output 0. Otherwise, output 1 iff the following equations are satisfied:
$$e(\overline{U}, g_1) = 1, \tag{8}$$
$$e\left(vk_{\mathcal{P}}, A_2\right) \cdot e\left(\prod_{j=1}^{T} h_j^{\overline{m}[j]}, A_1 \right) \cdot e(\overline{W}, A_1) = e(\overline{U}, g). \tag{9}$$

Theorem 10. *The scheme satisfies succinctness.*

Proof. The signature size of our HS scheme is independent of the dataset size.

Theorem 11. *The scheme satisfies signature correctness.*

Proof. The proof is the same as in theorem 6, by replacing R_i with τ_i in theorem 6.

Theorem 12. *The scheme satisfies evaluation correctness.*

Proof. The proof is the same as in theorem 7, by replacing R_i with τ_i in theorem 7.

Theorem 13. *The scheme satisfies unforgeability if DL and CDH problems are hard, F_K is a pseudorandom function, and the one-way function exists for adversary $\mathscr{B}$.*

Proof. To prove this theorem, we need to define a series of games with the adversary $\mathscr{B}$ and we will show that the adversary $\mathscr{B}$ wins (the game outputs 1) only with negligible probability.

- **Game 1**: This game is the security experiment $\mathsf{Exp}^{\mathsf{EUF-CMA}}_{\mathscr{B},\mathsf{HS}}$, where $\mathscr{B}$ only outputs forgery 1 and 2.
- **Game 2**: This game is defined as Game 1 apart from the fact that whenever $\mathscr{B}$ outputs a forgery tuple $(\mathcal{P}^*_\Delta, \overline{m}^*, \sigma^*)$, where $\sigma^* = (\sigma^*_\Delta, A^*, \overline{U}^*, \overline{W}^*)$ such that A^* was not generated by the challenger, then Game 2 sets $\mathsf{bad}_2 = \mathsf{true}$.
- **Game 3**: This game is defined as Game 2 except that the pseudorandom function F_K of our scheme is replaced with a true random function $F : \{0,1\}^* \to \mathbb{Z}_N$.
- **Game 4**: This game is defined as Game 3 except for an additional check. For a forgery tuple $(\mathcal{P}^*_\Delta, \overline{m}^*, \sigma^*)$, it computes $\overline{m} = \sum_{i=1}^{n} c_i \cdot m_i$, and checks whether $\prod_{j=1}^{T} h_j^{\overline{m}[j]} = \prod_{j=1}^{T} h_j^{\overline{m}^*[j]}$. If it does, sets $\mathsf{bad}_4 = \mathsf{true}$.

In Lemma 5 we show that Bad_4 is negligible for adversary $\mathscr{B}$ under DL assumption. Afterward, in Lemma 6, we show how a challenger can use an adversary $\mathscr{B}$ winning Game 4 to break the CDH assumption.

Lemma 5. $\mathsf{Pr}[\mathsf{Bad}_4] \leq \eta(\lambda)$ *if the DL assumption holds in G_1.*

Proof. The proof is the same as lemma 2.

Lemma 6. *The challenger can use an adversary $\mathscr{B}$ that wins in Game 4 to break the CDH assumption.*

Proof. Given $(bgpp = (N, G_1, G_2, G_T, e, g_1, g_2), g_1^a, g_1^b, g_2^b, g^b)$, where $a, b \xleftarrow{\$} \mathbb{Z}_N$, we show how the challenger to break CDH assumption in G_1.

– Setup: Let Q be the number of datasets in which the adversary $\mathscr{B}$ makes signature queries. The challenger guesses the dataset $\hat{\Delta}$ in which the adversary $\mathscr{B}$ produces a forgery, then runs $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$ except for the generation of the h. It choose $x_1, \ldots, x_T \xleftarrow{\$} \mathbb{Z}_N$, sets $h_j = (g_1^a)^{x_j}$ for $j \in [T]$. This is perfectly indistinguishable from an honest setup since x_j is random. Finally, the challenger gives pk to the adversary $\mathscr{B}$.
– Sign queries: The adversary $\mathscr{B}$ asks for signatures on (L, m), where $L = (\Delta, \tau)$.
  1. If the adversary $\mathscr{B}$ queries signatures for the dataset $\Delta \neq \hat{\Delta}$. The challenger runs $\alpha \leftarrow \mathsf{F}_K(\Delta)$, sets $A = (A_1, A_2) = (g^\alpha, g_2^\alpha)$, and $\sigma_\Delta \leftarrow \mathsf{Sign}'(sk', \Delta|A)$. It selects $r \xleftarrow{\$} \mathbb{Z}_N$, sets $W = g_1^r \cdot g_1^{a \cdot \sum_{j=1}^T -x_j \cdot m[j]}$, computes
$$U = \left(f_1(H(\tau)) \cdot \prod_{j=1}^{T} h_j^{m[j]} \cdot W \right)^\alpha = (f_1(H(\tau)) \cdot g_1^r)^\alpha .$$
  Finally, the challenger returns $\sigma = (\sigma_\Delta, A, U, W)$ to $\mathscr{B}$. The signature correctness can be easily verified.
  2. If the adversary $\mathscr{B}$ queries signatures for the dataset $\Delta = \hat{\Delta}$. The challenger selects $y \xleftarrow{\$} \mathbb{Z}_N$, then sets $A = (A_1, A_2) = ((g^b)^y, (g_2^b)^y)$ and $\sigma_\Delta \leftarrow \mathsf{Sign}'(sk', \Delta|A)$. It selects $r \xleftarrow{\$} \mathbb{Z}_N$, sets $W = f_1(H(\tau))^{-1} \cdot g_1^r \cdot g_1^{a \cdot \sum_{j=1}^T -x_j \cdot m[j]}$, computes $U = (g_1^b)^{r \cdot y}$. Finally, the challenger returns $\sigma = (\sigma_\Delta, A, U, W)$ to $\mathscr{B}$. The signature correctness can be verified: $e(U, g_1) = 1$ and
$$\begin{aligned}
&e(H(\tau), A_2) \cdot e\left(\prod_{j=1}^{T} h_j^{m[j]}, A_1 \right) \cdot e(W, A_1) \\
&= e(H(\tau), g_2^{b \cdot y}) \cdot e\left(f_1(H(\tau))^{-1} \cdot g_1^r \cdot g_1^{a \cdot \sum_{j=1}^T -x_j \cdot m[j]}, g^{b \cdot y} \right) \\
&\cdot e\left(\prod_{j=1}^{T} ((g_1^a)^{x_j})^{m[j]}, g^{b \cdot y} \right) \\
&= e(f_1(H(\tau)), g)^{b \cdot y} \cdot e(f_1(H(\tau))^{-1} \cdot g_1^r, g)^{b \cdot y} \\
&= e\left((g_1^b)^{r \cdot y}, g \right) \\
&= e(U, g).
\end{aligned}$$

– Forgery: The adversary $\mathscr{B}$ finally returns a forgery tuple $(\mathcal{P}_\Delta^*, \overline{m}^*, \sigma^*)$, where $\prod_{j=1}^T h_j^{\overline{m}[j]} \neq \prod_{j=1}^T h_j^{\overline{m}^*[j]}$ and $\sum_{j=1}^T x_j \cdot \overline{m}[j] \neq \sum_{j=1}^T x_j \cdot \overline{m}^*[j]$ since $\mathsf{bad}_4 = \mathsf{true}$.

$\overline{U}$ and $\overline{U}^*$ are valid signatures for the function f, we have

$$\begin{aligned}\overline{U} &= \left(\prod_{i=1}^{n} f_1(H(\tau_i))^{c_i} \cdot \overline{W}\right)^{b\cdot y} \cdot \left(\prod_{j=1}^{T} h_j^{\overline{m}[j]}\right)^{b\cdot y} \\ &= \left(\prod_{i=1}^{n} f_1(H(\tau_i))^{c_i}\right)^{b\cdot y} \cdot \left(\prod_{i=1}^{n} W_i^{c_i}\right)^{b\cdot y} \cdot \left(g_1^{b\cdot y\cdot a\cdot \sum_{j=1}^{T} x_j\cdot \overline{m}[j]}\right)\end{aligned}$$

and

$$\begin{aligned}\overline{U}^* &= \left(\prod_{i=1}^{n} f_1(H(\tau_i))^{c_i} \cdot \overline{W}\right)^{b\cdot y} \cdot \left(\prod_{j=1}^{T} h_j^{\overline{m}^*[j]}\right)^{b\cdot y} \\ &= \left(\prod_{i=1}^{n} f_1(H(\tau_i))^{c_i}\right)^{b\cdot y} \cdot \left(\prod_{i=1}^{n} W_i^{c_i}\right)^{b\cdot y} \cdot \left(g_1^{b\cdot y\cdot a\cdot \sum_{j=1}^{T} x_j\cdot \overline{m}^*[j]}\right).\end{aligned}$$

If compute $\overline{U}\cdot(\overline{U}^*)^{-1}$, we can get

$$\begin{aligned}\frac{\overline{U}}{\overline{U}^*} &= \frac{(\prod_{i=1}^{n} f_1(H(\tau_i))^{c_i})^{b\cdot y}\cdot(\prod_{i=1}^{n} W_i^{c_i})^{b\cdot y}\cdot\left(g_1^{b\cdot y\cdot a\cdot \sum_{j=1}^{T} x_j\cdot \overline{m}[j]}\right)}{(\prod_{i=1}^{n} f_1(H(\tau_i))^{c_i})^{b\cdot y}\cdot(\prod_{i=1}^{n} W_i^{c_i})^{b\cdot y}\cdot\left(g_1^{b\cdot y\cdot a\cdot \sum_{j=1}^{T} x_j\cdot \overline{m}^*[j]}\right)} \\ &= g_1^{a\cdot b\cdot(y\cdot\sum_{j=1}^{T} x_j\cdot(\overline{m}[j]-\overline{m}^*[j]))}.\end{aligned}$$

The challenger, therefore, can get g_1^{ab} by computing

$$\left(\frac{\overline{U}}{\overline{U}^*}\right)^{\frac{1}{y\cdot\sum_{j=1}^{T} x_j\cdot(\overline{m}[j]-\overline{m}^*[j])}} = g_1^{\frac{a\cdot b\cdot(y\cdot\sum_{j=1}^{T} x_j\cdot(\overline{m}[j]-\overline{m}^*[j]))}{y\cdot\sum_{j=1}^{T} x_j\cdot(\overline{m}[j]-\overline{m}^*[j])}} = g_1^{ab}.$$

5 Conclusion

In this work, we proposed a new primitive called ATPRG, which can be regarded as a new type of PRG. We showed how to construct a secure ATPRG via PRG, reverse re-randomizable encryption, and TBG. We further studied the applications of our ATPRG, and we used ATPRG to construct the first homomorphic signature scheme with only $O(T)$-sized public keys in the standard model.

Next, we discuss some other potential applications of ATPRG. (1) For data compression, if we use PRGen to generate the pseudorandom data (e.g., id), then we can use $\mathsf{rec}_{\mathscr{B}}$ to compress these data to $O(1)$ size, which even maybe close to the Shannon limit (traditional data compression algorithms are hard to compress pseudorandom data). Further, if we use SRGen to generate the pseudorandom data, then we can obtain a privacy-preserving data compression approach since only the users with the secret trapdoor can recover the data correctly; (2) for

information hiding, if we use PRGen and SRGen to generate two sequences of the pseudorandom data, we can obtain an information hiding approach, which means that the $\overline{pr}$ sequence is superficial data, but there hides a secret sequence $\overline{sr}$; (3) in [24], the authors use TBG to design an extended variant of the Boneh-Goh-Nissim cryptosystem, a variant of the Groth-Ostrovsky-Sahai NIZK, and new anonymous IBE, signature, and encryption schemes; therefore, our ATPRG may also be applicable in these fields.

We finally leave some open problems:

In which applications the discussional approaches are feasible?

and

How close is discussion (1) to the Shannon limit in different applications?

References

1. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. In: Rabin, T. (ed.) CRYPTO 2010. pp. 209–236. Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_12
2. Attrapadung, N., Libert, B., Peters, T.: Computing on authenticated data: New privacy definitions and constructions. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. pp. 367–385. Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_23
3. Backes, M., Fiore, D., Reischuk, R.M.: Verifiable delegation of computation on outsourced data. In: ACM CCS 2013. pp. 863–874 (2013). https://doi.org/10.1145/2508859.2516681
4. Boneh, D., Rubin, K., Silverberg, A.: Finding composite order ordinary elliptic curves using the cocks-pinch method. Journal of Number Theory **131**(5), 832–841 (2011). https://doi.org/10.1016/j.jnt.2010.05.001
5. Boneh, D., Freeman, D.M.: Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. pp. 1–16. Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19379-8_1
6. Catalano, D., Fiore, D.: Practical homomorphic macs for arithmetic circuits. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. pp. 336–352. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_21
7. Catalano, D., Fiore, D., Nizzardo, L.: Programmable hash functions go private: Constructions and applications to (homomorphic) signatures with shorter public keys. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. pp. 254–274. Springer, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_13
8. Catalano, D., Fiore, D., Warinschi, B.: Adaptive pseudo-free groups and applications. In: Paterson, K.G. (ed.) EUROCRYPT 2011. pp. 207–223. Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_13
9. Catalano, D., Marcedone, A., Puglisi, O.: Authenticating computation on groups: New homomorphic primitives and applications. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. pp. 193–212. Springer, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_11

10. Chen, W., Lei, H., Qi, K.: Lattice-based linearly homomorphic signatures in the standard model. Theoretical Computer Science **634**, 47–54 (2016). https://doi.org/10.1016/j.tcs.2016.04.009
11. Degabriele, J.P., Paterson, K.G., Schuldt, J.C.N., Woodage, J.: Backdoors in pseudorandom number generators: Possibility and impossibility results. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. pp. 403–432. Springer, Berlin, Heidelberg (2016). https://doi.org/10.1016/j.jnt.2010.05.001
12. Dodis, Y., Ganesh, C., Golovnev, A., Juels, A., Ristenpart, T.: A formal treatment of backdoored pseudorandom generators. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. pp. 101–126. Springer, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_5
13. Dolatnezhad Samarin, S., Fiore, D., Venturi, D., Amini, M.: A compiler for multi-key homomorphic signatures for turing machines. Theoretical Computer Science **889**, 145–170 (2021). https://doi.org/10.1016/j.tcs.2021.08.002
14. Fiore, D., Mitrokotsa, A., Nizzardo, L., Pagnin, E.: Multi-key homomorphic authenticators. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. pp. 499–530. Springer, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_17
15. Freeman, D.M.: Improved security for linearly homomorphic signatures: A generic framework. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. pp. 697–714. Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30057-8_41
16. Gennaro, R., Wichs, D.: Fully homomorphic message authenticators. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. pp. 301–320. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-42045-0_16
17. Gorbunov, S., Vaikuntanathan, V., Wichs, D.: Leveled fully homomorphic signatures from standard lattices. In: STOC 2015. pp. 469–477 (2015). https://doi.org/10.1145/2746539.2746576
18. Hébant, C., Phan, D.H., Pointcheval, D.: Linearly-homomorphic signatures and scalable mix-nets. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020. pp. 597–627. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45388-6_21
19. Hemenway, B., Libert, B., Ostrovsky, R., Vergnaud, D.: Lossy encryption: Constructions from general assumptions and efficient selective opening chosen ciphertext security. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. pp. 70–88. Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_4
20. Johnson, R., Molnar, D., Song, D., Wagner, D.: Homomorphic signature schemes. In: Preneel, B. (ed.) CT-RSA 2002. pp. 244–262. Springer, Berlin, Heidelberg (2002). https://doi.org/10.1007/3-540-45760-7_17
21. Lamport, L.: Constructing digital signatures from a one way function. Tech. Rep. CSL-98 (October 1979), https://www.microsoft.com/en-us/research/publication/constructing-digital-signatures-one-way-function/
22. Libert, B., Peters, T., Joye, M., Yung, M.: Linearly homomorphic structure-preserving signatures and their applications. Designs, Codes and Cryptography **77**(2), 441–477 (2015). https://doi.org/10.1007/s10623-015-0079-1
23. Luo, F., Wang, F., Wang, K., Chen, K.: A more efficient leveled strongly-unforgeable fully homomorphic signature scheme. Information Sciences **480**, 70–89 (2019). https://doi.org/10.1016/j.ins.2018.12.025

24. Meiklejohn, S., Shacham, H.: New trapdoor projection maps for composite-order bilinear groups. Cryptology ePrint Archive, Paper 2013/657 (2013), https://eprint.iacr.org/2013/657
25. Rompel, J.: One-way functions are necessary and sufficient for secure signatures. In: STOC 1990. pp. 387–394 (1990)
26. Schabhüser, L., Buchmann, J., Struck, P.: A linearly homomorphic signature scheme from weaker assumptions. In: O'Neill, M. (ed.) IMA International Conference on Cryptography and Coding (IMACC 2017). pp. 261–279. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71045-7_14
27. Schabhüser, L., Butin, D., Buchmann, J.: Context hiding multi-key linearly homomorphic authenticators. In: Matsui, M. (ed.) CT-RSA 2019. pp. 493–513. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-12612-4_25
28. Vazirani, U.V., Vazirani, V.V.: Trapdoor pseudo-random number generators, with applications to protocol design. In: SFCS 1983. pp. 23–30 (1983). https://doi.org/10.1109/SFCS.1983.78