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

Jinpeng Hou[1,2], Yansong Gao[3], Anmin Fu[1,2] (✉), Jie Chen[4], Xiaofeng Chen[2], Yuqing Zhang[5], Willy Susilo[6], and Josef Pieprzyk[3]

[1] School of Cyber Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China
`{jinpenghou,fuam}@njust.edu.cn`

[2] State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an 710071, China
`xfchen@xidian.edu.cn`

[3] Data61, CSIRO, Sydney, Australia
`{garrison.gao,josef.pieprzyk}@data61.csiro.au`

[4] School of Software Engineering, East China Normal University, Shanghai 200062, China
`s080001@e.ntu.edu.sg`

[5] National Computer Network Intrusion Protection Center, University of Chinese Academy of Sciences, Beijing, China
`zhangyq@ucas.ac.cn`

[6] School of Computing and Information Technology, Institute of Cybersecurity and Cryptology, University of Wollongong, Wollongong, NSW 2522, Australia
`wsusilo@uow.edu`

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 $pr_1, \dots, pr_N$ for the users having no knowledge of the public trapdoor and the secret trapdoor; so 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 this function is the same as backdoor pseudorandom generators. Further, the users having the secret trapdoor can use pr sequence to generate a sequence $sr_1, \dots, sr_N$ of the secret pseudorandom numbers. 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.

⋆ 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).

As for applications of ATPRG, we construct the first homomorphic signature scheme (in the standard model) whose public key size is only $O(T)$ that is 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 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 · Homomoprhic 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 Things (IoT) devices. To the best of our knowledge, only Catalano et al. (CRYPTO'15) [6] 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 [21] allow a remote party to compute any function 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, and 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 publicly verified. In the past two decades, HS has undergone great development in the computed function, especially from linearly HS that only supports addition or multiplication [1,6,8,9,14,15,19,21–23,26], to bounded polynomials [2,3,5,7] and to (level) fully homomorphic operations that allow computing the general arithmetic circuits of a priori bounded depth [12,16,17]. However, as a notable limitation, in the standard model, the public key sizes of [1–3,5,7–9,12,14,15,17,19,21–23,26] are all restricted to be linear with the dataset size; the smallest of [6] 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) [10,11,27], 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 $\mathsf{td}_{\mathscr{B}}$ [10]. $\mathscr{B}$ 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 the 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 (RRRE), 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. For a random element pr_i of elliptic curve group G, we 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). We use $sr_i \leftarrow f_1(pr_i)$ represents the operation that maps the public pseudorandom number pr_i to the secret pseudorandom number sr_i. 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 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.

However, the ATPRG construction described above is insufficient to solve the problem of constructing cryptographic schemes under the standard model since it is difficult to simulate the schemes without using the random oracle. Therefore, we further propose a variant of ATPRG called (ζ, N)-simulated ATPRG. This variant adds three new algorithms $\mathsf{init}', \mathsf{PRGen}', \mathsf{SRGen}'$ to the original ATPRG. We can initialize another simulated ATPRG using setup and init', which makes it feasible to use PRGen' to generate a simulated public pseudorandom sequence $\overline{pr}' = (pr'_1, \ldots, pr'_N)$, then use the SRGen' to generate a simulated secret pseudorandom sequence $\overline{sr}' = (sr'_1, \ldots, sr'_N)$. Where $\overline{sr}'$ has the ζ elements (we use sr'_ℓ to represent them) generated without using the secret trapdoor (but the equation $sr'_\ell = f_1(pr'_\ell)$ also holds), and $\mathsf{rec}_{\mathscr{B}}$ can still recover the entire $\overline{pr}'$ sequence from any element in $\overline{pr}'$. The requirement that generates ζ elements of $\overline{sr}'$ without using the secret trapdoor but $sr'_\ell = f_1(pr'_\ell)$ also holds may sound counter-intuitive. In a nutshell, we do this by using some of the underlying mathematical properties of the unusual bilinear map to inversely construct the corresponding pr'_ℓ from a random sr'_ℓ such that $sr'_\ell = f_1(pr'_\ell)$ holds simultaneously. We construct a $(1, N)$-simulated ATPRG instance again upon techniques of PRG, RRRE, and trapdoor projection maps for bilinear groups.

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 [6] proposed a HS scheme with $O(\sqrt{N}+\sqrt{T})$-sized public keys exploiting techniques of programmable hash and cover-free. We leverage $(1, N)$-simulated 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 $m_1, \ldots, m_N$, we use PRGen algorithm to generate N pseudorandom numbers $pr_1, \ldots, pr_N$, then use SRGen algorithm to transform $pr_1, \ldots, pr_N$ to $sr_1, \ldots, sr_N$. A simplified structure of our signature U_i is as following (the signature U_i is actually the secret pseudorandom number sr_i):

$$U_i = f_1 \left(pr_i \cdot \prod_{j=1}^{T} h_j^{m_i[j]} \right),$$

where $m_i[j]$ represents the j-th component of the i-th message, $h_1, \ldots, h_T$ are randomly selected from group G, and f_1 is the trapdoor projection map for $G \to G_1$. 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 $pr_1, \ldots, pr_N$ through an unusual bilinear mapping. In our HS scheme, $pr_1, \ldots, pr_N$ and $h_1, \ldots, h_T$ are the public keys, we only need to store one element pr_i for any $i \in [N]$ since the algorithm $\mathsf{rec}_{\mathscr{B}}$ can use pr_i and public trapdoor $\mathsf{td}_{\mathscr{B}}$ to recover the whole sequence $pr_1, \ldots, pr_N$. Therefore, the public key size of our scheme is $O(T)$. In the security proof, we use the algorithms $(\mathsf{setup}, \mathsf{init}', \mathsf{PRGen}', \mathsf{SRGen}', \mathsf{rec}_{\mathscr{B}})$ to simulate the operations of $(\mathsf{setup}, \mathsf{init}, \mathsf{PRGen}, \mathsf{SRGen}, \mathsf{rec}_{\mathscr{B}})$ such that we can cancel the use of the random oracle. Finally, we prove the unforgeability of our HS scheme based on the discrete logarithm (DL) assumption and the security property (real non-reversibility) of the (ζ, N)-simulated ATPRG. We summarize and compare the public key sizes of different schemes in Table 1.

ATPRG can help design more space-efficient protocols where data/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	Assumptions
[1–3, 5, 7–9, 12, 14, 15, 17, 19, 21–23, 26]	$O(N+T)$	-
[6]	$O(\sqrt{N}+\sqrt{T})$	q-DHI and FDHI
Our scheme	$O(T)$	DL and f_1-SGH

1.3 Organization

In Section 2, we introduce necessary notations and definitions. In Section 3, we give the definitions of ATPRG and (ζ, N)-simulated ATPRG, and construct the instances of them upon techniques of PRG, RRRE, and trapdoor projection maps for bilinear groups. In Section 4, we elaborate on the construction of our HS scheme with shorter public keys. In Section 5, we discuss other ATPRG applications and leave some open problems.

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 [11] 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*

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

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 [20] 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*

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

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 scheme has an additional property, reverse re-randomizable [10], compared to the re-randomizable encryption scheme.

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 [10,11], 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* $(n, l) \in \mathbb{N}^2$*:*

- $(pp, bk) \leftarrow \mathsf{setup}(coins) : \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$*. 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,* $bk = \perp$*.*
- $s \leftarrow \mathsf{init}(pp, coins) : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^n$*. 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.*
- $(r, s') \leftarrow \mathsf{next}(pp, s) : \{0,1\}^* \times \{0,1\}^n \to \{0,1\}^l \times \{0,1\}^n$*. 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 $\mathsf{state}^k(\mathsf{next}) = s_1, \ldots, s_k$ 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:*

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

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* $\mathsf{Adv}^{\mathsf{PRG}.\mathrm{dist}}_{\mathscr{A},q} \leq \delta$.

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:*

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

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* $\mathsf{Adv}^{\mathsf{PRG}.\mathrm{fwd}}_{\mathscr{A},q} \leq \delta$.

2.4 Backdoor Pseudorandom Generators

Differing from the standard PRG, there is another backdoor attacker $\mathscr{B}$ in BPRG; that is, there exist 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 [11], BPRG has three security advantages for $\mathscr{B}$: $\mathsf{Adv}_{\mathscr{B},q}^{\mathsf{BPRG.dist}}$, $\mathsf{Adv}_{\mathscr{B},q}^{\mathsf{BPRG.next}}$, and $\mathsf{Adv}_{\mathscr{B},q}^{\mathsf{BPRG.rsk}}$. And then [10] gives two stronger security advantages $\mathsf{Adv}_{\mathscr{B},q}^{\mathsf{BPRG.first}}$ and $\mathsf{Adv}_{\mathscr{B},q}^{\mathsf{BPRG.out}}$. We only give the definition of $\mathsf{Adv}_{\mathscr{B},q}^{\mathsf{BPRG.out}}$ since it implies the correctness of our ATPRG to recover public pseudorandom sequence. The original paper [10] of BPRG does not provide a direct definition of correctness; instead, it uses $\mathsf{Adv}_{\mathscr{B},q}^{\mathsf{BPRG.out}}$ to imply the correctness advantage, probably because the authors are targeting "Big Brother" to research.

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

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

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

Definition 11 ($(t, q, \delta, (\cdot, \epsilon))$-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* $\mathsf{Adv}_{\mathscr{B},q}^{\mathsf{BPRG}.(\cdot)} \geq \epsilon$ *and* $(\mathsf{setup}, \mathsf{init}, \mathsf{next})$ *is a* (t, q, δ)*-secure* PRG*.*

Definition 12 ($(t, q, \delta, (\cdot, \epsilon))$-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* $\mathsf{Adv}_{\mathscr{B},q}^{\mathsf{BPRG}.(\cdot)} \geq \epsilon$ *and* $(\mathsf{setup}, \mathsf{init}, \mathsf{next})$ *is a* (t, q, δ)-FWD*-secure* PRG*.*

2.5 Trapdoor Projection Maps for Bilinear Groups

Since this type of bilinear mapping is unusual, we will cover some of the details in a little more detail, and they will be used in the following constructions and proofs.

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 been 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

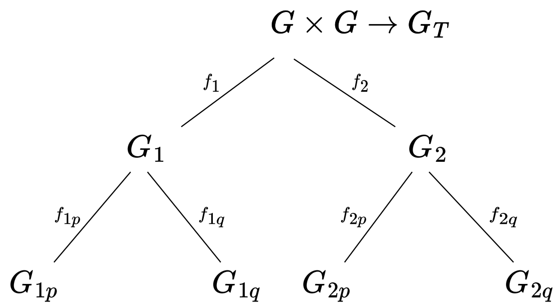

Fig. 1: TBG construction

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 13. $\mathsf{TBG.setup}(1^\lambda)$. *Use the algorithms in [24, Algorithm 1] to generate a prime q, a composite Q, an elliptic curve E defined over $\mathbb{F}_q$ such that $G := E[Q]$ contains Q^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 $G_T := \mu_Q$. Output bilinear group public parameters $bgpp = (Q, G_1, G_2, G_T, e, P_1, P_2)$.*

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, $bgpp = (Q, G_1, G_2, G_T, e, P_1, P_2)$ can be represented as $bgpp = (Q, G_1, G_2, G_T, e, g_1, g_2)$. Note that the group G generated by the $\mathsf{TBG.setup}(1^\lambda)$ algorithm has a fact that $G = G_1 \oplus G_2$; in other words, G has exponent Q but contains Q^2 points, it can be represented as the product of two cyclic groups (G_1 and G_2) of order Q.

Definition 14 (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}($\mathsf{td}_{\mathscr{S}} = f_{2q}$). Note that trapdoor projection map preserves the group structure, e.g., for an element $u \in G$ and its inverse $u^{-1} \in G$, $f_1(u)^{-1} = f_1(u^{-1})$. In addition, we have the following Lemma 1 to guarantee the quality of the above mappings.

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

For $\forall u \in G$, we have a unique decomposition $u = g_1^a \cdot g_2^b$. The computation of f_1 map is $f_1(u) = g_1^a$. Therefore, if we first select two random element $Y \in G_1$ and $Z \in G_2$, then compute $u = Y \cdot Z$. Finally, we have $f_1(u) = Y$. This is an important property to construct our $(1, N)$-simulated ATPRG. In addition, the following properties of the *bgpp* with trapdoor $\mathsf{td}_{\mathscr{S}} = f$ are also 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.

The following two assumptions related to the f_1 group are also involved in the proofs of this paper.

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

Assumption 2 (Decisional f_1-SGH) *Given bgpp* $\leftarrow \mathsf{TBG.setup}$, *random* $T \in G_T$, *and random elements* $u, v \xleftarrow{\$} G$ *for* $G : G_1 \times G_2$, *it is hard to tell if* $T = e(f_1(u), f_2(v))$ *or* T *is random [24, Assumption 4.5].*

2.6 Homomorphic Signatures

Definition 15 (Homomorphic Signatures). *A* HS *scheme has four PPT algorithms* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Ver})$*:*

- $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$. *Input a security parameter* λ. *Output a secret key* sk, *a public key* pk.
- $\sigma \leftarrow \mathsf{Sign}(sk, m)$. *Input the secret key* sk, *a message* m. *Output a signature* σ.
- $\overline{\sigma} \leftarrow \mathsf{Eval}(pk, f, \vec{\sigma})$. *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 all* $i \in [N]$. *Output an evaluation signature* $\overline{\sigma}$.
- $0/1 \leftarrow \mathsf{Ver}(pk, f, \overline{m}, \overline{\sigma})$. *Input the public key* pk, *a computed result* $\overline{m}$ *with its evaluation signature* $\overline{\sigma}$, *and the computed function* f *corresponding to* $\overline{\sigma}$. *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 16 (Signature Correctness). *A* HS *scheme satisfies signature correctness if for any key pair* $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and any signature* $\sigma \leftarrow \mathsf{Sign}(sk, m_i)$ *for all* $i \in [N]$*,* $1 \leftarrow \mathsf{Ver}(pk, f, m_i, \sigma_i)$ *holds with all but negligible probability.*

Definition 17 (Evaluation Correctness). *A* HS *scheme satisfies evaluation correctness if for any key pair* $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$*, any set of* $\{m_i, \sigma_i\}_{i=1}^N$ *and any function* $f : \mathcal{M}^N \to \mathcal{M}$ *such that* $1 \leftarrow \mathsf{Ver}(pk, f, \overline{m}, \overline{\sigma})$*, where* $\mathcal{M}$ *is the message space,* $\overline{m} \leftarrow f(m_1, \ldots, m_N)$*, and* $\overline{\sigma} \leftarrow \mathsf{Eval}(pk, f, \sigma_1, \ldots, \sigma_N)$*. In addition, this property also requires correctness for composed evaluation of several different functions. For any* $g_1, \ldots, g_\ell$ *with* $g_i : \mathcal{M}^N \to \mathcal{M}$ *and any* $f : \mathcal{M}^\ell \to \mathcal{M}$ *defines the composition* $(f \circ \bar{g}) : \mathcal{M}^N \to \mathcal{M}$ *by* $(f \circ \bar{g})(m) = h(g_1(m) \ldots, g_\ell(m))$*, any* $(m_1, \ldots, m_N) \in \mathcal{M}^N$*, any signature* $(\sigma_1, \ldots, \sigma_N)$ *such that* $1 \leftarrow \mathsf{Ver}(pk, g_i, m_i, \sigma_i)$ *for* $i \in [\ell]$*, and* $\overline{\sigma} \leftarrow \mathsf{Eval}(pk, f, \sigma_1, \ldots, \sigma_\ell)$ *computed on* f*, we have* $1 \leftarrow \mathsf{Ver}(pk, (f \circ \bar{g}), f(m_1, \ldots, m_\ell), \overline{\sigma})$*.*

Definition 18 (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 $\mathsf{Exp}_{\mathscr{A},\mathsf{HS}}^{\mathsf{EUF-CMA}}$ first.

Definition 19. *The experiment* $\mathsf{Exp}_{\mathscr{A},\mathsf{HS}}^{\mathsf{EUF-CMA}}$ *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* m*. If* m *is the first query, then the challenger initializes an empty list* $T = \emptyset$*. If* $m \notin T$*, the challenger runs* $\sigma \leftarrow \mathsf{Sign}(sk, m)$*, gives* σ *to* $\mathscr{A}$*, and adds* (m, σ) *to* T*. If* $m \in T$*, the challenger gives the corresponding signature* σ *to* $\mathscr{A}$*.*
- Forgery*: The adversary* $\mathscr{A}$ *finally returns a forgery tuple* $(f^*, \overline{m}^*, \overline{\sigma}^*)$*. The output of the experiment is 1 iff the following hold: (1)* f *is admissible on the message* $m_1, \ldots, m_N$*; (2)* $\overline{m}^* \neq \overline{m}$*; (3)* $1 \leftarrow \mathsf{Ver}(pk, f^*, \overline{m}^*, \overline{\sigma}^*)$*.*

Definition 20 (Security (Unforgeability)). *A* HS *scheme satisfies unforgeability if the following equation holds.*

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

Selective security: The selective security game requires that the adversary $\mathscr{A}$ sends all messages to the challenger for signature query before receiving the public key. Since the problem of shortening homomorphic signature public keys is difficult, we only provide security proof under the standard model in the selective security game.

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

Definition 21 (Efficient Verification). *A* HS *scheme for multi-labeled programs satisfies efficient verification, if there exists two algorithms* $(\mathsf{VerPerp}, \mathsf{EffVer})$ *such that:*

- $vk \leftarrow \mathsf{VerPerp}(pk, f)$*. Input the public key* pk *and the computed function* f*. Output a concise verification key* vk*.*
- $0/1 \leftarrow \mathsf{EffVer}(vk, \overline{m}, \overline{\sigma})$*. Input the concise verification key* vk *and a computed result* $\overline{m}$ *with its evaluation signature* $\overline{\sigma}$*. Output 0 (reject) or 1 (accept).*

$\mathsf{VerPerp}$ *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* $(f, \overline{m}, \overline{\sigma})$ *such that* $1 \leftarrow \mathsf{Ver}(pk, f, \overline{m}, \overline{\sigma})$*, for every* $vk \leftarrow \mathsf{VerPerp}(pk, f)$*, we have* $\mathsf{Pr}[\mathsf{EffVer}(vk, \overline{m}, \overline{\sigma}) = 1] = 1$*.*
2. ***Amortized Efficiency****. Given* f *and valid input messages* $m_1, \ldots, m_N$*, let* $t(N)$ *be the time of compute* $f(m_1, \ldots, m_N)$*, we require that the time of* $\mathsf{EffVer}(vk, \overline{m}, \overline{\sigma})$ *is independent of* N*, i.e.,* $O(1)$*.*

3 Asymmetric Trapdoor Pseudorandom Generators

3.1 Definitions

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

Definition 22. *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* $\overline{sr}$*. In addition, this algorithm also defines the public pseudorandom number space* G *and the secret pseudorandom number space* G_1*.*
- $s_0 \leftarrow \mathsf{init}(pp)$*. Output an initial state of* ATPRG*.*
- $\overline{pr} \leftarrow \mathsf{PRGen}(pp, s_0)$*. Input a public parameter* pp *and a initial state* s_0*. Output a public pseudorandom sequence* $\overline{pr} = (pr_1, \ldots, pr_N)$*, where* $pr_i \in G$ *for all* $i \in [N]$*.*
- $\overline{sr} \leftarrow \mathsf{SRGen}(pp, \overline{pr}, \mathsf{td}_{\mathscr{S}})$*. Input a public parameter* pp*, a public pseudorandom sequence* $\overline{pr}$*, and a secret trapdoor* $\mathsf{td}_{\mathscr{S}}$*. Output a secret pseudorandom sequence* $\overline{sr} = (sr_1, \ldots, sr_N)$*, where* $sr_i \in G_1$ *for all* $i \in [N]$*.*
- $\overline{pr} \leftarrow \mathsf{rec}_{\mathscr{B}}(pp, \mathsf{td}_{\mathscr{B}}, pr_N, N)$*. 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.*

We refer to the above ATPRG definition as the standard ATPRG definition to distinguish it from the variant definition below.

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. We expect ATPRG to be equivalent to a standard PRG for the users having no knowledge of the public trapdoor and the secret trapdoor, and ATPRG to be equivalent to a standard BPRG for the uses only having the public trapdoor.

An ATPRG should satisfy four properties: pseudorandomness, PR correctness, SR correctness, and non-reversibility. Where the non-reversibility is also the security of the standard ATPRG. Pseudorandomness requires that the public pseudorandom sequence is pseudorandom and the secret pseudorandom sequence is pseudorandom without the knowledge of the secret trapdoor. 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 $\mathsf{Adv}^{\mathsf{BPRG.out}}_{\mathscr{B},q} = 1$. 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. The formal definitions of these four properties are as follows (instead of giving a formal definition of pseudorandomness, we use a short description instead since it is very intuitive):

Definition 23 (ATPRG Inverse Advantage). *The inverse advantage of all PPT adversaries $\mathscr{A}$ is defined as follows:*

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

Definition 24 (ATPRG Correctness Advantage). *The correctness advantage of all PPT algorithms $\mathscr{C}$ is defined as follows:*

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

Definition 25 (Pseudorandomness). *An* ATPRG *satisfies pseudorandomness if:*

1. *The public pseudorandom sequence is pseudorandom;*
2. *Given the public pseudorandom sequence, the secret pseudorandom is also pseudorandom.*

Note that in the application of this paper, we do not force the second requirement of pseudorandomness to be true. That is, given the pseudorandom public sequence, the secret sequence does not have to be pseudorandom since our HS scheme is actually a unique signature. We retain this requirement to provide a basis for further applied research.

Definition 26 (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 27 (SR Correctness). *An* ATPRG *satisfies SR correctness if the advantage* $\mathsf{Adv}^{\mathsf{ATPRG}.\mathrm{crt}}_{\mathscr{C},q}$ *holds with overwhelming probability.*

Definition 28 (Non-Reversibility (Security)). *An* ATPRG *satisfies non-reversibility if* $\mathsf{Adv}^{\mathsf{ATPRG}.\mathrm{inv}}_{\mathscr{A},q}$ *is negligible.*

3.2 A Simple ATPRG Construction

We can use a (t, q, ϵ)-FWD-secure PRG $(\mathsf{setup}, \mathsf{init}, \mathsf{next})$, a (t, q, δ, ν)-IND\$-CPA-secure RRRE $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand}, \mathsf{Rand}^{-1})$, and TBG to construct an instance $\mathsf{ATPRG} = (\mathsf{setup}, \mathsf{init}, \mathsf{PRGen}, \mathsf{SRGen}, \mathsf{rec}_{\mathscr{B}})$. The details are shown in Fig.2. To simplify the description of the construction, we ignore the consistency adjustment of parameters between different primitives and algorithms, e.g., we assume that $pr_i \in G$ for all $i \in [N]$. In practice, we can adjust different parameters by using techniques such as one-way trapdoor function.

It is easy to see that the cryptographic scheme constructed based on the above ATPRG can be easily proved under the random oracle model, and the security depends on the non-reversibility[7] of ATPRG. However, this paper attempts to solve the long-standing open problem of constructing homomorphic signature schemes with shorter public key lengths under the standard model. Although the above simple ATPRG instance can construct the HS scheme, it is difficult to prove its security since, without the use of hash queries, $\overline{pr}$ compressed by ATPRG is difficult to simulate the operation of the HS scheme. In other words, the compressibility of $\overline{pr}$ makes it easy to construct various short public key schemes, but this compressibility also makes it difficult to simulate the schemes in security proof under the standard model. Therefore, a novel variant of ATPRG called (ζ, N)-simulated ATPRG is proposed to solve the above problem. We cover the construction of this variant in detail in the next section.

In addition, the ATPRG instance in Fig.2 uses RRRE, which is directly migrated from the BPRG. However, the BPRG uses RRRE to make it impossible for an adversary to tell whether the PRG has a backdoor $\mathsf{td}_{\mathscr{B}}$. Our ATPRG exposes $\mathsf{td}_{\mathscr{B}}$ so that users can use it to recover $\overline{pr}$. Therefore, we can actually remove the use of RRRE and simply set the initial state s_0 to be the public trapdoor. However, the use of RRRE can make ATPRG more flexible. It can not only generate $pr_1 \ldots, pr_N$ from the initial state but also generate them from

[7] The non-reversibility requirement can be adjusted appropriately according to different cryptographic schemes because this property is the security of ATPRG, and the security of ATPRG will affect the security of the applications constructed by ATPRG.

$\mathsf{setup}:$
$(pk, sk) \leftarrow \mathsf{RRRE.KeyGen}$;
$pp' \leftarrow \mathsf{PRG.setup}$;
$\mathsf{td}_{\mathscr{B}} \leftarrow sk$;
$(bgpp, \mathsf{td}_{\mathscr{S}}) \leftarrow \mathsf{TBG.setup}$;
$pp \leftarrow (pk, pp', bgpp)$;
Output $(pp, \mathsf{td}_{\mathscr{B}}, \mathsf{td}_{\mathscr{S}})$.

$\mathsf{init}(pp):$
$(pk, pp') \leftarrow pp$;
$s_0^* \leftarrow \mathsf{PRG.init}(pp')$;
$r_0 \leftarrow \mathsf{RRRE.Enc}(pk, s_0^*)$;
$s_0 \leftarrow (r_0, s_0^*)$;
Output s_0.

$\mathsf{PRGen}(pp, s_0):$
$(pk, pp') \leftarrow pp$;
$(r_0, s_0^*) \leftarrow s$;
$pr_0 \leftarrow r_0$;
$(t_1, \ldots, t_N) \leftarrow \mathsf{out}^N(\mathsf{PRG.next}(pp', s_0^*)$;
for $j = 1, \ldots, N$:
  $pr_j \leftarrow \mathsf{RRRE.Rand}(pr_{j-1}, t_j)$;
Output $\overline{pr} = (pr_1, \ldots, pr_N)$.

$\mathsf{SRGen}(pp, \overline{pr}, \mathsf{td}_{\mathscr{S}}):$
$bgpp \leftarrow pp$;
$(pr_1, \ldots, pr_N) \leftarrow \overline{pr}$;
$f_1 \leftarrow \mathsf{td}_{\mathscr{S}}$;
for $j = 1, \ldots, N$:
  $sr_j \leftarrow f_1(pr_j)$;
Output $\overline{sr} = (sr_1, \ldots, sr_N)$.

$\mathsf{rec}_{\mathscr{B}}(pp, \mathsf{td}_{\mathscr{B}}, pr_N, N):$
$sk \leftarrow \mathsf{td}_{\mathscr{B}}$;
$pp' \leftarrow pp$;
$s_0^* \leftarrow \mathsf{RRRE.Dec}(sk, pr_N)$;
$(t_1, \ldots, t_N) \leftarrow \mathsf{out}^N(\mathsf{PRG.next}(pp', s_0^*))$;
for $i = N, \ldots, 2$:
  $pr_{i-1} \leftarrow \mathsf{RRRE.Rand}^{-1}(pr_N, t_N)$;
Output $\overline{pr} = (pr_1, \ldots, pr_N)$.

Fig. 2: A standard ATPRG construction

an optional position i for $i \in [N]$, which is crucial for the construction of our (ζ, N)-simulated ATPRG below. Therefore, we do not directly delete the use of RRRE here. If the subsequent studies only construct and prove the security of cryptographic schemes using this standard ATPRG, we recommend removing RRRE for efficiency.

Next, we briefly demonstrate the properties of the ATPRG instance in Fig.2.

Theorem 1. *The* ATPRG *instance satisfies pseudorandomness.*

Proof. It is easy to see that the public sequence $\overline{pr}$ is pseudorandom. The pseudorandomness of the secret sequence is dependent on Assumption 2.

Theorem 2. *The* ATPRG *instance satisfies PR correctness.*

Proof. From [10], the algorithms $(\mathsf{setup}, \mathsf{init}, \mathsf{PRGen}, \mathsf{rec}_{\mathscr{B}})$ of the ATPRG instance form a $(t, q, \delta, (out, 1))$-FWD-secure BPRG.

Theorem 3. *The* ATPRG *instance satisfies SR correctness.*

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

Theorem 4. *The* ATPRG *instance satisfies non-reversibility.*

Proof. In a nutshell, Assumption 1 shows that, for non-reversibility advantage, $\nexists i \in [q]$, $sr^*_i \leftarrow \mathscr{B}(pp, pr_i)$ such that $sr_i = sr^*_i$. Therefore, $\mathsf{Adv}^{\mathsf{ATPRG.inv}}_{\mathscr{C},q} = \eta(\lambda)$. We can easily prove the theorem in the random oracle model.

3.3 (ζ, N)-Simulated ATPRG

Definition 29. *A* (ζ, N)*-simulated ATPRG has eight algorithms* (setup, init, init', PRGen, PRGen', SRGen, SRGen', $\mathsf{rec}_{\mathscr{B}}$)*. Where* ($\mathsf{setup}$, init, PRGen, SRGen, $\mathsf{rec}_{\mathscr{B}}$) *is a standard ATPRG, the definitions of them are the same as Definition 22 except that the* SRGen *adds an input of a message sequence* $\overline{M} = (M_1, \ldots, M_N)$*.* ($\mathsf{setup}$, init', PRGen', SRGen', $\mathsf{rec}_{\mathscr{B}}$) *called the simulated ATPRG. The definitions of* init', PRGen', SRGen', SRGen *are different from the standard ATPRG, the details are as follows:*

- $s_0 \leftarrow \mathsf{init}'(pp)$*. Output an initial state of the simulated* ATPRG*.*
- $\overline{pr}' \leftarrow \mathsf{PRGen}'(pp, s_0)$*. Output a simulated public pseudorandom sequence* $\overline{pr}' = (pr'_1, \ldots, pr'_N)$ *such that* $\overline{pr}' \leftarrow \mathsf{rec}_{\mathscr{B}}(pp, \mathsf{td}_{\mathscr{B}}, pr'_N, N)$ *and* $pr'_i \in G$ *for all* $i \in [N]$*.*
- $\overline{sr}' \leftarrow \mathsf{SRGen}'(pp, \overline{pr}', \mathsf{td}_{\mathscr{S}}, \overline{M})$*. Input a public parameter pp, a simulated public pseudorandom sequence* $\overline{pr}'$*, a secret trapdoor* $\mathsf{td}_{\mathscr{S}}$*, and a message sequence* $\overline{M} = (M_1, \ldots, M_N)$*. Output a simulated secret pseudorandom sequence* $\overline{sr}' = (sr'_1, \ldots, sr'_N)$*, where* $sr'_i \in G_1$ *for all* $i \in [N]$*,* ζ *elements of* $\overline{sr}'$ *are generated without using the secret trapdoor* $\mathsf{td}_{\mathscr{S}}$ *and other elements are generated by* $\mathsf{td}_{\mathscr{S}}$*.*
- $\overline{sr} \leftarrow \mathsf{SRGen}(pp, \overline{pr}, \mathsf{td}_{\mathscr{S}}, \overline{M})$*. Input a public parameter pp, a public pseudorandom sequence* $\overline{pr}$*, a secret trapdoor* $\mathsf{td}_{\mathscr{S}}$*, and a message sequence* $\overline{M} = (M_1, \ldots, M_N)$*. Output a secret pseudorandom sequence* $\overline{sr} = (sr_1, \ldots, sr_N)$*, where* $sr_i \in G_1$ *for all* $i \in [N]$*.*

We call the sequence $\overline{pr} \leftarrow \mathsf{PRGen}$ as real public pseudorandom sequence and the sequence $\overline{pr}' \leftarrow \mathsf{PRGen}'$ as simulated public pseudorandom sequence (correspondingly, $\overline{sr} \leftarrow \mathsf{SRGen}$ is real secret pseudorandom sequence and $\overline{sr}' \leftarrow \mathsf{SRGen}'$ is simulated secret pseudorandom sequence). We require that (setup, init, PRGen, SRGen, $\mathsf{rec}_{\mathscr{B}}$) and ($\mathsf{setup}$, init', PRGen', SRGen', $\mathsf{rec}_{\mathscr{B}}$) have the following properties: pseudorandomness, PR correctness, SR correctness, and non-reversibility. That is, both have the standard ATPRG function.

Specifically, a (ζ, N)-simulated ATPRG has eight properties: real pseudorandomness, simulated pseudorandomness, real PR correctness, simulated PR correctness, real SR correctness, simulated SR correctness, real non-reversibility, and simulated non-reversibility. The definitions associated with "real" are the properties of algorithms (setup, init, PRGen,SRGen, $\mathsf{rec}_{\mathscr{B}}$) and the definitions associated with "simulated" are the properties of algorithms (setup, init', PRGen',

$\mathsf{SRGen}', \mathsf{rec}_{\mathscr{B}}$). Where real and simulated pseudorandomness and real and simulated PR correctness are the same as the standard ATPRG, the real and simulated SR correctness requires that the algorithm $\mathscr{C}$ also knows the message sequence $\overline{M}$. The real and simulated non-reversibility are different from the standard ATPRG, we have a stronger requirement. Specifically, let $m_i = (m_i[1], \ldots, m_i[T])$ for all $i \in [N]$ are the T-dimension vectors, given $\overline{pr} = (pr_1, \ldots, pr_N)$ (resp. $\overline{pr}' = (pr'_1, \ldots, pr'_N)$), $\overline{sr} = (sr_1, \ldots, sr_N)$ (resp. $\overline{sr}' = (sr'_1, \ldots, sr'_N)$), $h_j \in G$ for all $j \in [T]$, $\overline{M} = \left(\prod_{j=1}^{T} h_j^{m_1[j]}, \ldots, \prod_{j=1}^{T} h_j^{m_N[j]}\right) = (M_1, \ldots, M_N)$[8], it should be difficult to find $m_k^* \neq m_k$, $\prod_{j=1}^{T} h_j^{m_k^*[j]} \neq \prod_{j=1}^{T} h_j^{m_k[j]}$ and $sr_k^* \in G_1$ for $k \in [N]$, such that $f_1\left(pr_k \cdot \prod_{j=1}^{T} h_j^{m_k^*[j]}\right) = sr_k^*$.[9] (Note that the condition that $m_k^* \neq m_k$, $\prod_{j=1}^{T} h_j^{m_k^*[j]} = \prod_{j=1}^{T} h_j^{m_k[j]}$ is not holds under the DL assumption, the details can be found in Lemma 2.) This requirement may be a little elusive, and the readers can further understand its meaning by the following HS security definition; in other words, the real non-reversibility guarantees the security of our HS scheme.

The definitions of the eight properties are as follows (to reduce redundancy, we combine two similar definitions to describe them).

Definition 30 (Real (resp. Simulated) Pseudorandomness). *A (ζ, N)-simulated* ATPRG *satisfies real (resp. simulated) pseudorandomness if:*

1. *The real (resp. simulated) public pseudorandom sequence is pseudorandom;*
2. *Given the real (resp. simulated) public pseudorandom sequence, the real (resp. simulated) secret pseudorandom is also pseudorandom.*

Definition 31 (Real (resp. Simulated) PR Correctness). *A (ζ, N)-simulated* ATPRG *satisfies real (resp. simulated) PR correctness if the standard (resp. simulated) ATPRG is the $(t, q, \delta, (out, 1))$-secure n-outputs* BPRG *or $(t, q, \delta, (out, 1))$-*FWD*-secure n-outputs* BPRG*.*

Definition 32 (Real SR Correctness). *A (ζ, N)-simulated* ATPRG *satisfies real SR correctness if*

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

---

8 $\prod_{j=1}^{T} h_j^{m_i[j]}$ is some algebraic hash function. This paper restricts the format of messages to such hash function since not all formats of messages satisfy this security requirement.

9 In the (ζ, N)-simulated ATPRG, we add a message sequence, and we change the computation of SRGen from $sr_i = f_1(pr_i)$ to $sr_i = f_1\left(pr_i \cdot \prod_{j=1}^{T} h_j^{m_i[j]}\right)$ (resp. $sr'_i = f_1\left(pr'_i \cdot \prod_{j=1}^{T} h_j^{m_i[j]}\right)$) for all $i \in [N]$.

holds with overwhelming probability for all PPT algorithms $\mathscr{C}$*.*

Definition 33 (Simulated SR Correctness). *A* (ζ, N)*-simulated* ATPRG *satisfies simulated SR correctness if* $\mathsf{Adv}_{\mathscr{C},q}^{\mathsf{simATPRG}'.\mathrm{crt}}$ *holds with overwhelming probability for all PPT algorithms* $\mathscr{C}$*. Where* $\mathsf{Adv}_{\mathscr{C},q}^{\mathsf{simATPRG}'.\mathrm{crt}}$ *is obtained by replacing the algorithms* $(\mathsf{init}, \mathsf{PRGen}, \mathsf{SRGen})$ *in* $\mathsf{Adv}_{\mathscr{C},q}^{\mathsf{simATPRG}.\mathrm{crt}}$ *with the simulated ATPRG algorithms* $(\mathsf{init}', \mathsf{PRGen}', \mathsf{SRGen}')$*. Therefore,* $sr_1', \ldots, sr_q' \leftarrow \mathsf{SRGen}'$ *have* ζ *elements that are generated without using* $\mathsf{td}_{\mathscr{S}}$*. This property implies that the* ζ *simulated secret pseudorandom elements that are generated without using* $\mathsf{td}_{\mathscr{S}}$ *also have the property (correctness) of the real secret pseudorandom numbers, which means that they can also use the correspondingly simulated public pseudorandom numbers and* $\mathsf{td}_{\mathscr{S}}$ *to compute correctly.*

Definition 34 (Real Non-Reversibility). *A* (ζ, N)*-simulated* ATPRG *satisfies real non-reversibility if*

$$\mathsf{Adv}_{\mathscr{A},N}^{\mathsf{simATPRG}.\mathrm{inv}} := \Pr\left[\begin{array}{c} (pp, \mathsf{td}_{\mathscr{B}}, \mathsf{td}_{\mathscr{S}}) \leftarrow \mathsf{setup}, \\ m_1, \ldots, m_N \stackrel{\$}{\leftarrow} \{0,1\}^*, \\ h_1, \ldots, h_T \stackrel{\$}{\leftarrow} \{0,1\}^*, \\ s_0 \leftarrow \mathsf{init}(pp), \\ \overline{pr} = (pr_1, \ldots, pr_N) \leftarrow \mathsf{PRGen}, \\ \overline{sr} = (sr_1, \ldots, sr_N) \leftarrow \mathsf{SRGen}, \\ \exists i \in [N], sr_i^*, m_i^* \leftarrow \mathscr{A}(pp, \mathsf{td}_{\mathscr{B}}, \\ \overline{pr}, \overline{sr}, m_1, \ldots, m_N, h_1, \ldots, h_T) : \\ m_i \neq m_i^*, \prod_{j=1}^T h_j^{m_i[j]} \neq \prod_{j=1}^T h_j^{m_i^*[j]}, \\ sr_i^* = \mathsf{SRGen}(pp, pr_i, \mathsf{td}_{\mathscr{S}}, \prod_{j=1}^T h_j^{m_i^*[j]}) \end{array}\right]$$

holds with negligible probability for all PPT adversaries $\mathscr{A}$*, where* $m_i[j] \in \mathbb{Z}_Q$ *for all* $i \in [N], j \in [T]$*, and* $h_j \in G$ *for all* $j \in [T]$*.*[10]

Definition 35 (Simulated Non-Reversibility). *A* (ζ, N)*-simulated* ATPRG *satisfies simulated non-reversibility if* $\mathsf{Adv}_{\mathscr{A},N}^{\mathsf{simATPRG}'.\mathrm{inv}}$ *is negligible, where the advantage* $\mathsf{Adv}_{\mathscr{A},N}^{\mathsf{simATPRG}'.\mathrm{inv}}$ *is obtained by replacing the algorithms* $(\mathsf{init}, \mathsf{PRGen}, \mathsf{SRGen})$ *in* $\mathsf{Adv}_{\mathscr{A},N}^{\mathsf{simATPRG}.\mathrm{inv}}$ *with the simulated ATPRG algorithms* $(\mathsf{init}', \mathsf{PRGen}', \mathsf{SRGen}')$*. In other words, we require that the pair* (sr_i', pr_i') *(*sr_i *is generated without using* pr_i *and* $\mathsf{td}_{\mathscr{S}}$*) and the pair* (sr_j', pr_j') *(*sr_j *is generated using* pr_j *and* $\mathsf{td}_{\mathscr{S}}$*) are have the same "security".*

On the other hand, we actually use $(\mathsf{setup}, \mathsf{init}, \mathsf{PRGen}, \mathsf{SRGen}, \mathsf{rec}_{\mathscr{B}})$ to generate $(\overline{pr}, \overline{sr})$ to construct the homomorphic signatures in the real scheme, whereas we use $(\mathsf{setup}, \mathsf{init}', \mathsf{PRGen}', \mathsf{SRGen}', \mathsf{rec}_{\mathscr{B}})$ to generate $(\overline{pr}', \overline{sr}')$ to simulate the HS scheme in the security proof. Therefore, the four "simulated" properties are intended to correspond to the four "real" properties, and their purpose is to help

[10] $sr_i^* = \mathsf{SRGen}(pp, pr_i, \mathsf{td}_{\mathscr{S}}, \prod_{j=1}^T h_j^{m_i^*[j]})$ actually means $sr_i^* = f_1\left(pr_i \cdot \prod_{j=1}^T h_j^{m_i^*[j]}\right)$.

the simulated scheme become indistinguishable from the real scheme in the security proof. In a nutshell, the simulated pseudorandomness and simulated PR correctness guarantee that the adversary $\mathscr{A}$ cannot distinguish the real scheme from the simulated scheme by the simulated public pseudorandom sequences. The simulated PR and SR correctness guarantee that the simulated signatures can also be verified correctly such that the adversary $\mathscr{A}$ cannot distinguish the real scheme from the simulated scheme by the simulated secret pseudorandom sequences.[11] The simulated non-reversibility guarantee that the simulated scheme has the same "security" as the real scheme, such that we can reduce the security of the real scheme to the real non-reversibility.[12] In other words, the adversary $\mathscr{A}$ distinguishes the real scheme from the simulated scheme by the flaws of "correctness" and "randomness", and can launch the useless attacks through some flaws in the simulated scheme [18]. The eight properties can help the challenger (which runs the simulated scheme) to guarantee that these actions of the adversary $\mathscr{A}$ are not feasible.

We give a construction[13] of the $(1, N)$-simulated ATPRG in Fig.3. That is, one element sr'_ℓ of the simulated secret pseudorandom sequence is generated without the use of secret trapdoor but $sr'_\ell = f_1(pr'_\ell \cdot M_\ell)$ holds. Note that $pr_i, M_i \in G, sr_i \in G_1$ for all $i \in [N]$.

Our new construction imposes some additional requirements on RRRE and PRG. Specifically, we require RRRE to have the following additional properties: strong reverse re-randomness and strong decryption randomness. Strong reverse re-randomness requires that, given a valid pseudorandom ciphertext and a pseudorandom number as inputs to RRRE's re-randomizable algorithm, the algorithm can output a valid and pseudorandom reverse ciphertext. Strong decrypt randomness requires that, given a valid pseudorandom ciphertext, the decrypt plaintext is also pseudorandom. It is easy to see that ElGamal encryption [13] satisfies all the requirements for RRRE in this paper. For PRG, we require that it can use a valid random value s'_0 as the initial state such that the algorithm $\mathsf{PRG.next}$ can also output pseudorandom string by s'_0. This requirement for PRG is nature since the initialization algorithm of PRG is essentially to produce a random initial state so that the process of producing a valid random value s'_0 is actually equivalent to the initialization algorithm init. Therefore, there are many

[11] The verification algorithm of our HS scheme needs to use $\mathsf{rec}_{\mathscr{B}}$ to recover the whole $\overline{pr}$ sequence. Thus, here needs not only SR correctness but also PR correctness.

[12] The main role of the simulated non-reversibility is to help the challenger to stop the adversary from launching useless attacks.

[13] The output state s_0 of init' in Fig.3 does not contain $r_0 \leftarrow \mathsf{RRRE.Enc}(pk, s_0^*)$ since it does not affect any of the ATPRG properties. Specifically, the input of $\mathsf{RRRE.Enc}$ algorithm implies a random coin; thus, the adversary cannot compute a r_0^* such that $r_0^* = r_0$ (that is, $pr_1 \leftarrow \mathsf{RRRE.Rand}(r_0^*, t_1)$) by s_0^* and $\mathsf{RRRE.Enc}$ algorithm. The adversary can only use $\mathsf{RRRE.Rand}^{-1}(pr_1, t_1)$ to obtain the r_0 such that $pr_1 \leftarrow \mathsf{RRRE.Rand}(r_0, t_1)$. In other words, if the adversary can compute a $r_0^* = r_0$, given a pr_1, it can distinguish the real scheme from the simulated scheme based on whether r_0^* equals $\mathsf{RRRE.Rand}^{-1}(pr_1, t_1)$, since the simulated public pseudorandom number pr'_1 maybe not holds that $\mathsf{RRRE.Rand}^{-1}(pr'_1, t_1) = r_0^*$.

PRGs that can satisfy this requirement, such as Dual EC PRG, which is studied in [10, 11]. Note that although the Dual EC PRG has a backdoor, it does not affect our use since our recovery algorithm $\mathsf{rec}_{\mathscr{B}}$ includes the step of recalculating the output of the PRG through the backdoor.

Definition 36 (Strong Reverse Re-Randomness of RRRE). *For all PPT adversaries $\mathscr{A}$, it holds that*

$$\left|\mathsf{Pr}\left[\begin{array}{c}(pk, sk) \leftarrow \mathsf{RRRE.KeyGen}, b \xleftarrow{\$} \{0,1\}, \\ C^* \xleftarrow{\$} \{0,1\}^*, C^0 \xleftarrow{\$} \{0,1\}^*, \\ t \xleftarrow{\$} \{0,1\}^*, C^1 \leftarrow \mathsf{RRRE.Rand}^{-1}(C^*, t), \\ b' \leftarrow \mathscr{A}(pk, sk, C^b): \\ b' = b\end{array}\right] - \frac{1}{2}\right| \leq \eta(\lambda),$$

where C^* and C^0 are valid ciphertexts.

Definition 37 (Strong Decryption Randomness of RRRE). *For all PPT adversaries $\mathscr{A}$, it holds that*

$$\left|\mathsf{Pr}\left[\begin{array}{c}(pk, sk) \leftarrow \mathsf{RRRE.KeyGen}, b \xleftarrow{\$} \{0,1\}, \\ C^* \xleftarrow{\$} \{0,1\}^*, M^0 \xleftarrow{\$} \{0,1\}^*, \\ M^1 \leftarrow \mathsf{RRRE.Dec}(sk, C^*), b' \leftarrow \mathscr{A}(pk, sk, M^b): \\ b' = b\end{array}\right] - \frac{1}{2}\right| \leq \eta(\lambda),$$

where C^* is a valid ciphertext and M^0 is a valid plaintext.

Next, we demonstrate the properties of $(1, N)$-simulated ATPRG.

Theorem 5. *The $(1, N)$-simulated* ATPRG *satisfies real and simulated pseudorandomness.*

Proof. Note that we do not force real and simulated secret pseudorandom sequences to be pseudorandom here, but it is clear that real and simulated secret sequences are still pseudorandom under the decisional f_1-SGH assumption. Similar to the Theorem 1, real pseudorandomness is obvious. The pseudorandomness of the simulated public sequence depends on the strong reverse re-randomness and strong decryption randomness of RRRE. Specifically, in $(1, N)$-simulated ATPRG, we randomly generate a simulated public pseudorandom number pr'_i, and then decrypt it to produce an initial state s'_0. The strong decryption randomness of RRRE guarantees that the initial state s'_0 is pseudorandom. Hence, $pr'_i, \ldots, pr'_N$ is obviously pseudorandom, which depends on the re-randomness of RRRE. For $pr'_1, \ldots, pr'_{i-1}$, we use the pseudorandom number generated by PRG and pr_i to calculate the Rand^{-1} algorithm for RRRE to generate them. Therefore, $pr'_1, \ldots, pr'_{i-1}$ is also pseudorandom if RRRE has strong reverse re-randomness.

Theorem 6. *The $(1, N)$-simulated* ATPRG *satisfies real and simulated PR correctness.*

$\underline{\mathsf{setup}:}$
$(pk, sk) \leftarrow \mathsf{RRRE.KeyGen};$
$pp' \leftarrow \mathsf{PRG.setup};$
$\mathsf{td}_{\mathscr{B}} \leftarrow sk;$
$(bgpp, \mathsf{td}_{\mathscr{S}}) \leftarrow \mathsf{TBG.setup};$
$pp \leftarrow (pk, pp', bgpp);$
Output $(pp, \mathsf{td}_{\mathscr{B}}, \mathsf{td}_{\mathscr{S}})$.

$\underline{\mathsf{init}(pp):}$
$(pk, pp') \leftarrow pp;$
$s_0^* \leftarrow \mathsf{PRG.init}(pp');$
$r_0 \leftarrow \mathsf{RRRE.Enc}(pk, s_0^*);$
$s_0 \leftarrow (r_0, s_0^*);$
Output s_0.

$\underline{\mathsf{init}'(pp, \mathsf{td}_{\mathscr{B}}, M_\ell, \ell):}$
$(pk', bgpp) \leftarrow pp;$
$sk \leftarrow \mathsf{td}_{\mathscr{B}};$
$Y \xleftarrow{\$} G_1;$
$Z \xleftarrow{\$} G_2;$
$sr'_\ell \leftarrow Y;$
$pr'_\ell \leftarrow Y \cdot Z \cdot M_\ell^{-1};$
$s_0^* \leftarrow \mathsf{RRRE.Dec}(sk, pr'_\ell);$
$s_0 \leftarrow s_0^*;$
Output s_0.

$\underline{\mathsf{PRGen}(pp, s_0):}$
$(pk, pp') \leftarrow pp;$
$(r_0, s_0^*) \leftarrow s_0;$
$pr_0 \leftarrow r_0;$
$(t_1, \ldots, t_N) \leftarrow \mathsf{out}^N(\mathsf{PRG.next}(pp', s_0^*);$
for $j = 1, \ldots, N:$
  $pr_j \leftarrow \mathsf{RRRE.Rand}(pr_{j-1}, t_j);$
Output $\overline{pr} = (pr_1, \ldots, pr_N)$.

$\underline{\mathsf{PRGen}'(pp, s_0, pr'_\ell, \ell):}$
$(pk, pp') \leftarrow pp;$
$s_0^* \leftarrow s'_0;$
$(t_1, \ldots, t_N) \leftarrow \mathsf{out}^N(\mathsf{PRG.next}(pp', s_0^*));$
for $j = \ell + 1, \ldots, N:$
  $pr'_j \leftarrow \mathsf{RRRE.Rand}(pr'_{j-1}, t_j);$
for $j = \ell - 1, \ldots, 1:$
  $pr'_j \leftarrow \mathsf{RRRE.Rand}^{-1}(pr'_{j+1}, t_{j+1});$
Output $\overline{pr}' = (pr'_1, \ldots, pr'_N)$.

$\underline{\mathsf{SRGen}(pp, \overline{pr}, \mathsf{td}_{\mathscr{S}}, \overline{M}):}$
$bgpp \leftarrow pp;$
$(pr_1, \ldots, pr_N) \leftarrow \overline{pr};$
$(M_1, \ldots, M_N) \leftarrow \overline{M};$
$f_1 \leftarrow \mathsf{td}_{\mathscr{S}};$
for $j = 1, \ldots, N:$
  $sr_j \leftarrow f_1(pr_j \cdot M_j);$
Output $\overline{sr} = (sr_1, \ldots, sr_N)$.

$\underline{\mathsf{SRGen}'(pp, \overline{pr}', \mathsf{td}_{\mathscr{S}}, \overline{M}, sr'_\ell, \ell):}$
$bgpp \leftarrow pp;$
$(pr'_1, \ldots, pr'_N) \leftarrow \overline{pr}';$
$(M_1, \ldots, M_N) \leftarrow \overline{M};$
$f_1 \leftarrow \mathsf{td}_{\mathscr{S}};$
for $j \neq \ell, j \in [N]:$
  $sr'_j \leftarrow f_1(pr'_j \cdot M_j);$
Output $\overline{sr}' = (sr'_1, \ldots, sr'_N)$.

$\underline{\mathsf{rec}_{\mathscr{B}}(pp, \mathsf{td}_{\mathscr{B}}, pr_N, N):}$
$sk \leftarrow \mathsf{td}_{\mathscr{B}};$
$pp' \leftarrow pp;$
$s_0^* \leftarrow \mathsf{RRRE.Dec}(sk, pr_N);$
$(t_1, \ldots, t_N) \leftarrow \mathsf{out}^N(\mathsf{PRG.next}(pp', s_0^*));$
for $i = N, \ldots, 2:$
  $pr_{i-1} \leftarrow \mathsf{RRRE.Rand}^{-1}(pr_N, t_N);$
Output $\overline{pr} = (pr_1, \ldots, pr_N)$.

Fig. 3: $(1, N)$-Simulated ATPRG

Proof. It is easy to see that the recovery algorithm $\mathsf{rec}_{\mathscr{B}}$ can still correctly recover the entire real and simulated public pseudorandom sequence. That is, the input of $\mathsf{rec}_{\mathscr{B}}$ in Fig.3 also can be $(pp, \mathsf{td}_{\mathscr{B}}, pr'_N, N)$.

Theorem 7. *The* $(1, N)$*-simulated* ATPRG *satisfies real and simulated SR correctness.*

Proof. Similar to Theorem 3, real SR correctness is obviously found to be true. We mainly analyze the simulated SR correctness. Let the simulated secret pseudorandom number be sr_i, obviously, for $j \neq i, j \in [N]$, we have

$$\mathsf{Pr}\begin{bmatrix} (pp, \mathsf{td}_{\mathscr{B}}, \mathsf{td}_{\mathscr{S}}) \leftarrow \mathsf{setup}, \\ s_0 \leftarrow \mathsf{init}', \\ pr'_1, \ldots, pr'_N \leftarrow \mathsf{PRGen}', \\ sr'_1, \ldots, sr'_N \leftarrow \mathsf{SRGen}', \\ sr^*_j \leftarrow \mathscr{C}(pp, \mathsf{td}_{\mathscr{S}}, pr_j) : \\ sr'_j = sr^*_j \end{bmatrix} = 1.$$

At the same time, we have $sr_i = Y, pr_i = Y \cdot Z \cdot M_i^{-1}$, where $Y \xleftarrow{\$} G_1, Z \xleftarrow{\$} G_2, M_i \in G$. Therefore, we have $f_1(pr_i \cdot M_i) = sr_i$.

Theorem 8. *The* $(1, N)$*-simulated* ATPRG *satisfies real and simulated non-reversibility if the* f_1*-SGH is hard.*

Proof. The proof of real non-reversibility and simulated non-reversibility are similar, and we need to prove both under the standard model. The proof of real non-reversibility is as follows. In a nutshell, assume that $\mathscr{A}$ is some PPT attacker that breaks the non-reversibility with non-negligible probability, given an element $w \in G$, we show that how the challenger break the f_1-SGH assumption (obtain $f_1(w)$ without using f_1). The challenger first selects two indexs $k \in [N], \ell \in [T]$ as the simulated index, chooses $h_j \xleftarrow{\$} G$ for $j \neq \ell, j \in [T]$ and $x_\ell \xleftarrow{\$} \mathbb{Z}_Q$, sets $h_\ell = w^{x_\ell}$. Then, it sets $sr_k = Y$ for $Y \xleftarrow{\$} G_1$, $pr_k = Y \cdot Z \cdot \prod_{j=1}^{T} h_j^{-m_k[j]}$ for $Z \xleftarrow{\$} G_2$, pr_i are pseudorandom numbers for $i \neq k, i \in [N]$, sr_i are $f_1\left(pr_i \cdot \prod_{j=1}^{T} h_j^{m_i[j]}\right)$ for $i \neq k, i \in [N]$. Then, the challenger sends $(pp, \mathsf{td}_{\mathscr{B}}, pr_1, \ldots, pr_N, sr_1, \ldots, sr_N, m_1, \ldots, m_N, h_1, \ldots, h_T)$ to the adversary $\mathscr{A}$. If the secret pseudorandom sequence and the message sequence returned by $\mathscr{A}$ contains $sr^*_k, m^*_k \neq m_k$ and m^*_k differs from m_k only in the ℓ-th dimension. We have

$$\frac{sr^*_k}{sr_k} = \frac{f_1\left(pr_k \cdot h_\ell^{m^*_k[\ell]}\right)}{f_1\left(pr_k \cdot h_\ell^{m_k[\ell]}\right)} = f_1(w)^{x_\ell \cdot (m^*_k[\ell] - m_k[\ell])}. \tag{2}$$

The challanger can compute

$$\left(\frac{sr^*_k}{sr_k}\right)^{\frac{1}{x_\ell \cdot \left(m^*_k[\ell] - m_k[\ell]\right)}} = f_1(w)$$

to obtain $f_1(w)$ without using the secret trapdoor $\mathsf{td}_{\mathscr{S}}$ to break the f_1-SGH assumption. Technically, the proof is still not complete at this point. We actually need N hybrid arguments to conclude the proof. In a nutshell, let the i-th argument select $k = i$ for $i \in [N]$, and the residual computation is the same as the procedure above. Note that the equation 2 involves a transformation that $f_1(w^{x_\ell}) = f_1(w)^{x_\ell}$, which is obviously true according to the details of Section 2.5.

In fact, the f_1-SGH assumption is relatively obscure, previous studies only provided the proof idea of this assumption under the random oracle model. Our work also provides the first proof idea under the standard model of this assumption, which can also lay a foundation for the further research.

4 Homomorphic Signature Scheme with Shorter Public Keys

We shows an approach constructing HS scheme using $(1, N)$-simulated ATPRG, which can provide design ideas for other cryptographic schemes based on ATPRG. Specifically, we generate secret pseudorandom sequence $\overline{sr} = (sr_1, \ldots, sr_N)$ by a public pseudorandom sequence $\overline{pr} = (pr_1, \ldots, pr_N)$ and a message sequence $m_1, \ldots, m_N$. In this construction, sr_i is the signature of message m_i and $\overline{pr}$ is public key. The public trapdoor $\mathsf{td}_{\mathscr{B}}$ and pr_i for any $i \in [n]$ can be used to recover the whole sequence $pr_1, \ldots, pr_n$. Note that the secret pseudorandom sequence does not have to be pseudorandom since the signature is unique.

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. $(pp, \mathsf{td}_{\mathscr{B}}, \mathsf{td}_{\mathscr{S}}) \leftarrow \mathsf{ATPRG.setup}$, then run $\mathsf{ATPRG.init}$ to initialize ATPRG;
  2. Randomly choose T elements $h_1, \ldots, h_T$ from G group, where T is the dimension of message m, let $h = (h_1, \ldots, h_T)$;
  3. Run $\mathsf{ATPRG.PRGen}$ to get $V_1, \ldots, V_N$ such that we have $V_1, \ldots, V_N \leftarrow \mathsf{rec}_{\mathscr{B}}(pp, \mathsf{td}_{\mathscr{B}}, V_N, N)$, where $V_i \in G$ for $i \in [N]$;
  4. $sk \leftarrow (pp, \mathsf{td}_{\mathscr{S}})$, $pk \leftarrow (pp, \mathsf{td}_{\mathscr{B}}, h, V_N)$.

- $\sigma_i \leftarrow \mathsf{Sign}(sk, m_i)$: To sign a message $m_i = (m_i[1], \ldots, m_i[T])$ for $m_i[j] \in \mathbb{Z}_Q, j \in [T]$, proceed as follows:
  1. Compute

$$U_i = f_1\left(V_i \cdot \prod_{j=1}^{T} h_j^{m_i[j]}\right) \leftarrow \mathsf{ATPRG.SRGen}\left(pp, V_i, \mathsf{td}_{\mathscr{S}}, \prod_{j=1}^{T} h_j^{m_i[j]}\right).$$

[14]Note that the ATPRG.SRGen algorithm we show in Fig.3 takes the entire public pseudorandom sequence $\overline{pr}$ and message sequence $\overline{M}$ as input and outputs the entire secret random sequence $\overline{sr}$ directly. However, it can also support the operation that takes one public pseudorandom number pr_i and one message m_i as input and outputs one corresponding secret pseudorandom number sr_i;

2. $\sigma_i \leftarrow U_i$.

- $\overline{\sigma} \leftarrow \mathsf{Eval}(pk, f, \vec{\sigma})$: for a linear function $f : \mathbb{Z}_Q^N \to \mathbb{Z}_Q$ described by its coefficients $f = (c_1, \ldots, c_N)$
  1. $(\sigma_1, \ldots, \sigma_N) \leftarrow \vec{\sigma}$;
  2. Compute
  $$\overline{U} = \prod_{i=1}^{N} U_i^{c_i};$$
  3. $\overline{\sigma} \leftarrow \overline{U}$.

- $0/1 \leftarrow \mathsf{Ver}(pk, f, \overline{m}, \overline{\sigma})$: for verifying the correctness of the computed result $\overline{m} = \sum_{i=1}^{N} c_i \cdot m_i$,
  1. Run $\mathsf{rec}_{\mathscr{B}}(pp, \mathsf{td}_{\mathscr{B}}, V_N, N)$ to recover $V_1, \ldots, V_N$;
  2. Output 1 iff the following equations are satisfied:
  $$e(\overline{\sigma}, g_1) = 1, \tag{3}$$
  $$e(\overline{\sigma}, g) = e\left(\prod_{j=1}^{T} h_j^{m_i[j]}, g_2\right) \cdot e\left(\prod_{i=1}^{N} V_i^{c_i}, g_2\right). \tag{4}$$

Note that: 1. the bilinear map of TBG is non-degenerate; hence, we need to use equation 3 first to check that the evaluation signature is still a member of G_1; 2. $\prod_{j=1}^{T} h_j^{m_i[j]}$ is some algebraic hash function.

Comparison of public key size: The public key of [6] is $(pk', bgpp, pek, pek')$, where pk' is the public key of the regularly digital signature scheme, $bgpp$ 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, [6] has $O(\sqrt{N} + \sqrt{T})$-sized public keys. The public key of our construction is $pk = (pp, \mathsf{td}_{\mathscr{B}}, h, V_N)$, where the size of $(pp, \mathsf{td}_{\mathscr{B}}, V_N)$ 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.

Theorem 9. *The scheme satisfies efficient verification.*

Proof. The verification algorithm can be rewritten as:

[14] To understand quickly, the readers can think of U_i as $\left(V_i \cdot \prod_{j=1}^{T} h_j^{m_i[j]}\right)^{\alpha}$ first; that is, to some extent, f_1 can be regarded as α. However, f_1 has more functions than α since f_1 also maps the element $V_i \cdot \prod_{j=1}^{T} h_j^{m_i[j]} \in G$ to the corresponding element $U_i \in G_1$.

– $vk \leftarrow \mathsf{VerPerp}(pk, f)$. Input the public key pk and the computed function f. Run $\mathsf{rec}_{\mathscr{B}}(pp, \mathsf{td}_{\mathscr{B}}, V_N, N)$ to recover $V_1, \ldots, V_N$, compute $vk = f(V_1, \ldots, V_N) = \prod_{i=1}^{N} V_i^{c_i}$. Output a concise verification key vk.
– $0/1 \leftarrow \mathsf{EffVer}(vk, \overline{m}, \overline{\sigma})$. Output 1 iff the following equations are satisfied:

$$e(\overline{\sigma}, g_1) = 1, \tag{5}$$

$$e(\overline{\sigma}, g) = e\left(\prod_{j=1}^{T} h_j^{m_i[j]}, g_2\right) \cdot e\left(vk, g_2\right). \tag{6}$$

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. Let $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$ be an honestly generated key pair with $(sk = (pp, \mathsf{td}_{\mathscr{S}}), pk = (pp, \mathsf{td}_{\mathscr{B}}, h, V_N))$ and let $\sigma_i \leftarrow \mathsf{Sign}(sk, m_i)$ be an honestly generated signature. For $\mathsf{Ver}(pk, f, m_i, \sigma_i)$, we have

$$e(\sigma_i, g_1) = 1,$$

since $\sigma_i = U_i \in G_1$. And

$$\begin{aligned} e(\sigma_i, g) &= e\left(\prod_{j=1}^{T} h_j^{m_i[j]}, g_2\right) \cdot e(V_i, g_2) \\ \Rightarrow e(U_i, g) &= e\left(\prod_{j=1}^{T} h_j^{m_i[j]} \cdot V_i, g_2\right) \\ \Rightarrow e\left(f_1\left(\prod_{j=1}^{T} h_j^{m_i[j]} \cdot V_i\right), g\right) &= e\left(\prod_{j=1}^{T} h_j^{m_i[j]} \cdot V_i, g_2\right) \\ \Rightarrow e\left(\prod_{j=1}^{T} h_j^{m_i[j]} \cdot V_i, g_2\right) &= e\left(\prod_{j=1}^{T} h_j^{m_i[j]} \cdot V_i, g_2\right). \end{aligned}$$

Theorem 12. *The scheme satisfies evaluation correctness.*

Proof. Let $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$ be an honestly generated key pair with $(sk = (pp, \mathsf{td}_{\mathscr{S}}), pk = (pp, \mathsf{td}_{\mathscr{B}}, h, V_N))$. Given $\{m_i, \sigma_i\}_{i=1}^{N}$ such that $1 \leftarrow \mathsf{Ver}(pk, f, m_i, \sigma_i)$, for all $i \in [N]$. Let $\overline{\sigma} \leftarrow \mathsf{Eval}(pk, f, \sigma_1, \ldots, \sigma_N)$. Finally, we want to prove that the verification algorithm $\mathsf{Ver}(pk, f, \overline{m}, \overline{\sigma})$ outputs 1.

Since each σ_i verifies correctly, we have $\overline{\sigma} = \prod_{i=1}^{N} U_i^{c_i} \in G_1$, thus $e(\overline{\sigma}, g_1) = 1$. And for equation 4, we have

$$
\begin{aligned}
&e(\overline{\sigma}, g) = e\left(\prod_{j=1}^{T} h_j^{\overline{m}[j]}, g_2\right) \cdot e\left(\prod_{i=1}^{N} V_i^{c_i}, g_2\right) \\
\Rightarrow\ & e\left(\prod_{i=1}^{N} U_i^{c_i}, g\right) = e\left(\prod_{i=1}^{N}\left(\prod_{j=1}^{T} h_j^{m_i[j]}\right)^{c_i} \cdot \prod_{i=1}^{N} V_i^{c_i}, g_2\right) \\
\Rightarrow\ & e\left(\prod_{i=1}^{N} f_1\left(\prod_{j=1}^{T} h_j^{m_i[j]} \cdot V_i\right)^{c_i}, g\right) = e\left(\prod_{i=1}^{N}\left(\prod_{j=1}^{T} h_j^{m_i[j]} \cdot V_i\right)^{c_i}, g_2\right) \\
\Rightarrow\ & e\left(\prod_{i=1}^{N}\left(\prod_{j=1}^{T} h_j^{m_i[j]} \cdot V_i\right)^{c_i}, g_2\right) = e\left(\prod_{i=1}^{N}\left(\prod_{j=1}^{T} h_j^{m_i[j]} \cdot V_i\right)^{c_i}, g_2\right).
\end{aligned}
$$

Theorem 13. *The scheme satisfies unforgeability if DL is hard and the $(1, N)$-simulated ATPRG has real non-reversibility.*

Proof. To prove this theorem, we need to define a series of games with the adversary $\mathscr{A}$ and we will show that the adversary $\mathscr{A}$ wins (the game outputs 1) only with negligible probability. Following the notation of [6], we also write $G_i(\mathscr{A})$ 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 $\mathsf{Exp}_{\mathscr{A},\mathsf{HS}}^{\mathsf{EUF-CMA}}$.
- **Game 2**: This game is defined as Game 1 except for an additional check. For a forgery tuple $(f^*, \overline{m}^*, \overline{\sigma}^*)$, 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 $\mathsf{bad}_2 = \mathsf{true}$.

Games 1 and 2 have the following relationship: $|\mathsf{Pr}[G_1(\mathscr{A})] - \mathsf{Pr}[G_2(\mathscr{A})]| \leq \mathsf{Pr}[\mathsf{Bad}_2]$. In Lemma 2 we show that Bad_2 is negligible for adversary $\mathscr{A}$ under DL assumption. In Lemma 3 we show how a challenger can use an adversary $\mathscr{A}$ winning Game 2 to break the real non-reversibility of the $(1, N)$-simulated ATPRG.

Lemma 2. $\mathsf{Pr}[\mathsf{Bad}_2] \leq \eta(\lambda)$ *if the DL assumption holds in G.*

Proof. Given $(g, g^a) \in G$, we show how the challenger to break DL assumption in G. Our simulation is similar to [25].

- 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 $x_1, \ldots, x_T \xleftarrow{\$} \mathbb{Z}_Q$, sets $h_j = g^{x_j}$ for $j \neq i$ and sets $h_i = (g^a)^{x_i}$. This is perfectly indistinguishable from a real execution of this game since x_j is random. Finally, the challenger gives pk to the adversary $\mathscr{A}$.

- Sign queries: The challenger answers all queries of the adversary $\mathscr{A}$ faithfully.
- Forgery: The adversary $\mathscr{A}$ finally returns a forgery tuple $(f^*, \overline{m}^*, \sigma^*)$, where $\prod_{j=1}^{T} h_j^{\overline{m}[j]} = \prod_{j=1}^{T} h_j^{\overline{m}^*[j]}$.

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

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

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

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 $\mathscr{A}$ that wins in Game 2 to break the real non-reversibility of the $(1, N)$-simulated ATPRG.*

Proof.
- Setup: The adversary $\mathscr{A}$ chooses the messages $m_1, \ldots, m_N$ to be signed and sends them to the challenger. The challenger guesses an index k from $[T]$, chooses $h_j \xleftarrow{\$} G$ for $j \neq k, j \in [T]$ and $x_k \xleftarrow{\$} \mathbb{Z}_Q$, sets $h_k = w^{x_k}$ for a given element $w \in G$. This is perfectly indistinguishable from an honest setup since x_k and h_j is random. Then, the challenger runs $(pp, \mathsf{td}_{\mathscr{B}}, \mathsf{td}_{\mathscr{S}}) \leftarrow \mathsf{ATPRG.setup}$, $\mathsf{ATPRG.init}'$, $\mathsf{ATPRG.PRGen}'$ of the $(1, N)$-simulated ATPRG in sequence. Finally, the challenger gives $pk = (pp, \mathsf{td}_{\mathscr{B}}, h, V'_N)$ to the adversary $\mathscr{A}$.
- Sign queries: The challenger answers all queries using the $\mathsf{ATPRG.SRGen}'$ of the $(1, N)$-simulated ATPRG, where the message m_ℓ and signature σ_ℓ is simulated. In other words, $U_\ell = Y$ for $Y \xleftarrow{\$} G_1$, $V_\ell = U_\ell \cdot Z \cdot \prod_{j=1}^{T} h_j^{-m_\ell[j]}$ for $Z \xleftarrow{\$} G_2$, V_i are pseudorandom numbers for $i \neq \ell, i \in [N]$, U_i are $f_1\left(V_i \cdot \prod_{j=1}^{T} h_j^{m_i[j]}\right)$ for $i \neq \ell, i \in [N]$.
- Forgery: The adversary $\mathscr{A}$ finally returns a forgery tuple $(f^*, \overline{m}^*, \sigma^*)$, where $\prod_{j=1}^{T} h_j^{\overline{m}[j]} \neq \prod_{j=1}^{T} h_j^{\overline{m}^*[j]}$.

For the forgery tuple $(f^* = (c_1, \ldots, c_N), \overline{m}^*, \sigma^*)$, the challenger checks whether only $\overline{m}[k] \neq \overline{m}^*[k]$. If not, restarts the simulation. Otherwise, we have

$$e(\overline{U}, g) \cdot e\left(\prod_{j=1}^{T} h_j^{-\overline{m}[j]}, g_2\right) = e(\overline{V}, g_2), \tag{7}$$

$$e(\overline{U}^*, g) \cdot e\left(\prod_{j=1}^{T} h_j^{-\overline{m}^*[j]}, g_2\right) = e(\overline{V}, g_2). \tag{8}$$

Therefore,

$$e(\overline{U}, g) \cdot e\left(\prod_{j=1}^{T} h_j^{-\overline{m}[j]}, g_2\right) = e(\overline{U}^*, g) \cdot e\left(\prod_{j=1}^{T} h_j^{-\overline{m}^*[j]}, g_2\right). \tag{9}$$

where $\overline{U} = \prod_{i=1}^{N} U_i^{c_i}$, $\overline{V} = \prod_{i=1}^{N} V_i^{c_i}$, $\overline{m} = c_1 \cdot m_1 + \cdots + c_N \cdot m_N$ and $\prod_{j=1}^{T} h_j^{\overline{m}[j]} = \prod_{j=1}^{T} h_j^{c_1 \cdot m_1[j] + \cdots + c_N \cdot m_N[j]}$. Let

$$U'_\ell = \left(\frac{\overline{U}^*}{\prod_{i=1,i\neq\ell}^{N} {U_i}^{c_i}}\right)^{-c_\ell},\ m'_\ell = \left(\frac{\overline{m}^*}{\sum_{i=1,i\neq\ell}^{N} c_i \cdot m_i}\right)/c_\ell.$$

We multiply the left-hand side of equation 7 and 8 by both $e\left(\frac{1}{\prod_{i=1,i\neq\ell}^{N} {U_i}^{c_i}}, g\right)$ and $e\left(\prod_{i=1,i\neq\ell}^{N}\left(\prod_{j=1}^{T} h_j^{m_i[j]}\right)^{c_i}, g_2\right)$, and then compute the whole thing to the power of $-c_\ell$:

$$\begin{aligned}
&(e\left(\frac{1}{\prod_{i=1,i\neq\ell}^{N} {U_i}^{c_i}}, g\right) \cdot e(\overline{U}, g) \cdot e\left(\prod_{i=1,i\neq\ell}^{N}\left(\prod_{j=1}^{T} h_j^{m_i[j]}\right)^{c_i}, g_2\right) \\
&\cdot e\left(\prod_{j=1}^{T} h_j^{-\overline{m}_i[j]}, g_2\right))^{-c_\ell} \\
&= \left(e(U_\ell^{c_\ell}, g) \cdot e\left(\left(\prod_{j=1}^{T} h_j^{-m_\ell[j]}\right)^{c_\ell}, g_2\right)\right)^{-c_\ell} \\
&= \left(\left(e\left(U_\ell \cdot f_1\left(\prod_{j=1}^{T} h_j^{-m_\ell[j]}\right), g\right)\right)^{c_\ell}\right)^{-c_\ell} \\
&= e\left(U_\ell \cdot f_1\left(\prod_{j=1}^{T} h_j^{-m_\ell[j]}\right), g\right) \\
&= e(f_1(V_\ell), g) \\
&= e(V_\ell, g_2),
\end{aligned} \tag{10}$$

$$
\begin{aligned}
&(e\left(\frac{1}{\prod_{i=1,i\neq\ell}^{N} U_i^{c_i}}, g\right)\cdot e(\overline{U}^{*}, g)\cdot e\left(\prod_{i=1,i\neq\ell}^{N}\left(\prod_{j=1}^{T} h_j^{m_i[j]}\right)^{c_i}, g_2\right)\\
&\cdot e\left(\prod_{j=1}^{T} h_j^{-\overline{m}_i^{*}[j]}, g_2\right))^{-c_\ell}\\
&= e\left(U'_\ell, g\right)\cdot e\left(\prod_{j=1}^{T} h_j^{-m'_\ell[j]}, g_2\right)\\
&= e\left(U'_\ell\cdot f_1\left(\prod_{j=1}^{T} h_j^{-m'_\ell[j]}\right), g\right).
\end{aligned}
\tag{11}
$$

From equations 9, 10, and 11, we have

$$
e\left(U_\ell\cdot f_1\left(\prod_{j=1}^{T} h_j^{-m_\ell[j]}\right), g\right) = e(f_1(V_\ell), g) = e\left(U'_\ell\cdot f_1\left(\prod_{j=1}^{T} h_j^{-m'_\ell[j]}\right), g\right).
$$

Therefore,

$$
U_\ell = f_1\left(V_\ell\cdot\prod_{j=1}^{T} h_j^{m_\ell[j]}\right),\ U'_\ell = f_1\left(V_\ell\cdot\prod_{j=1}^{T} h_j^{m'_\ell[j]}\right).
$$

It is easy to see that (U'_ℓ, m'_ℓ) is a solution of the real non-reversibility of $(1, N)$-simulated ATPRG, where $m'_\ell \neq m_\ell$. Technically, the proof also needs N hybrid arguments. In a nutshell, let the i-th argument select $\ell = i$ for $i \in [N]$, and the residual computation is the same as the procedure above.

5 Discussion

We first discuss some application scenarios where ATPRG might be used in addition to shortening the public key. (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 $\overline{sr}$.

Next, we leave some open problems. Firstly, this paper reduces only the public key of the linear homomorphic signature to $O(T)$. Whether it can be reduced to $O(1)$ by constructing a new ATPRG is still an open problem. Secondly, ATPRG in this paper is constructed based on an unusual symmetric bilinear map. If ATPRG can be constructed based on a common tool, it will have a better application prospect. Thirdly, the homomorphic signature scheme in this paper has only been proven in the selective security game, and further research into the full security game is a good choice. Fourthly, the ATPRG in this paper can only support linear homomorphic computation. Therefore, it is very useful to study the ATPRG that supports fully homomorphic computation and then construct the fully homomorphic signature with the shorter public key under the standard model. Finally, theoretically, the ATPRG proposed in this paper can try to shorten the public keys of various pairing-based cryptography schemes. Therefore, it is feasible to study the cryptography schemes with shorter public keys based on the standard ATPRG or the $(1, N)$-simulated ATPRG.

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., 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
7. 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
8. 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
9. 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

10. 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
11. 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
12. 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
13. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE transactions on information theory **31**(4), 469–472 (1985)
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. Guo, F., Susilo, W., Mu, Y.: Introduction to security reduction. Springer (2018)
19. 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
20. 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
21. 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
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. 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
26. 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
27. 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