AnoPas: Practical anonymous transit pass from group signatures with time-bound keys

Rui Shi [a], Yang Yang [b,*], Yingjiu Li [c], Huamin Feng [a], Hwee Hwa Pang [b], Robert H. Deng [b]

[a] Institute of Information Security, Beijing Electronic Science and Technology Institute, Beijing, 100070, China
[b] School of Computing and Information Systems, Singapore Management University, 188065, Singapore
[c] Department of Computer and Information Science, University of Oregon, Eugene, 97403, USA

Published in Journal of Systems Architecture (2024) 153. DOI: 10.1016/j.sysarc.2024.103184

Abstract: An anonymous transit pass system allows passengers to access transport services within fixed time periods, with their privileges automatically deactivating upon time expiration. Although existing transit pass systems are deployable on powerful devices like PCs, their adaptation to more user-friendly devices, such as mobile phones with smart cards, is inefficient due to their reliance on heavy-weight operations like bilinear maps. In this paper, we introduce an innovative anonymous transit pass system, dubbed Anopas, optimized for deployment on mobile phones with smart cards, where the smart card is responsible for crucial lightweight operations and the mobile phone handles key-independent and time-consuming tasks. Group signatures with time-bound keys (GS-TBK) serve as our core component, representing a new variant of standard group signatures for the secure use of time-based digital services, preserving users' privacy while providing flexible authentication services. We first constructed a practical GS-TBK scheme using the tag-based signatures and then applied it to the design of AnoPas. We achieve the most efficient passing protocol compared to the state-of-the-art AnoPas/GS-TBK schemes. We also present an implementation showing that our passing protocol takes around 38.6 ms on a smart card and around 33.6 ms on a mobile phone.

Keywords: Transit pass, Group signatures, Smart cards, Tag-based signatures, Efficient revocation

1. Introduction

The transit pass system refers to a comprehensive public transportation arrangement to improve urban mobility. It encompasses the provision of passes to passengers granting access to various modes of public transport service, including buses, trains, trams, and subways, within fixed time periods. These passes are available on different temporal scales (e.g., daily, weekly, monthly, or annually) and can offer unlimited or limited travel within the transit network. For example, the Navigo Pass [1] in France allows passengers to take unlimited trips only during weekends; the EZ transit Pass [2] in America is a monthly pass for local travel on 23 different public transit carriers. This system plays a pivotal role in promoting sustainable urban transportation by reducing private car usage, alleviating traffic congestion, and lessening environmental impacts. It also facilitates equitable access to transportation services, catering to the diverse needs of urban populations.

Anonymous Transit Pass. The anonymous transit pass [3], [4] system, as an enhanced version of the general transit pass, emphasizes minimizing collecting and retaining personally identifiable information about passengers. This approach prioritizes passenger privacy and addresses concerns about surveillance and data collection in public transportation systems. These passes can be used to access multiple modes of public transportation without linking usage data to specific individuals.

Despite the many achievements [3], [4] in anonymous transit pass system research, deployment in mobile application scenarios still faces unresolved issues. One such issue is the inability to deploy anonymous transit pass systems in resource-constrained devices, such as smart cards. Smart card devices are the most suitable platform for deploying transit passes due to their advantages of low power consumption, secure storage, and high portability. Existing anonymous transit pass systems [3], [4], [5], [6], [7], [8], [9] can be used in powerful devices such as personal computers. However, their schemes use time-consuming operations that are not supported by current standard smart card devices, so they cannot be used in lightweight devices. Secondly, preventing passing keys from being shared among unauthorized users is impossible. An important issue in the implementation of anonymous transit pass systems is the inability to prevent unpaid (unauthorized) devices (users) from accessing transport services. The most effective solution to this problem is to store the passing keys in a secure hardware device (such as a

smart card with secure storage and anti-copying capabilities), so that the transit pass protocol cannot be completed without the participation of the secure hardware. This approach not only addresses the issue of unrestricted shared passing keys by malicious users but also provides an additional layer of security for honest users.

While the smart card device has numerous advantages, it lacks a complete power supply and communication device. As a result, it requires external helper equipment to complete reading, writing, and computing operations through contact or non-contact. Therefore, implementing the anonymous transit pass protocol on the client side of the smart card without relying on helper devices is impossible. Mobile phones are the optimal choice for helper devices due to their widespread deployment, high frequency of use, abundant external interfaces, and efficient computing and communication capabilities.

Group signatures with time-bound keys. Group signature, introduced by Chaum and van Heyst [5], is a primary cryptography tool for user authentication, which is anonymous yet accountable to users and certifiable to services. Group signatures with time-bound keys (GS-TBK) were proposed by Chu et al. [6], as a new variant of standard group signatures allowing users to show a group signature to a server in some time periods, and their signing rights are automatically revoked once time expires. Another feature is that users' signing rights are invalid if the group manager prematurely revokes their signing keys before their expiry times have passed. GS-TBK retains all excellent features of standard group signatures, such as anonymity, traceability, and revocability.

Based on the transit pass system's functions, the GS-TBK scheme can be trivially converted into an anonymous transit pass system. However, while existing GS-TBK schemes [6–9] can be implemented on powerful devices such as embedded devices, mobile phones, or PCs, they are not efficient enough for deployment on smart cards because they rely on bilinear map operations, which are still unsupported on smart cards. To address this issue, in this paper, we propose a novel GS-TBK scheme with strong security features, which is significantly more efficient than the state-of-the-art, and further use it to construct an anonymous transit pass system named AnoPas, that can be deployed on mobile phones with smart cards.

1.1. Contribution

We introduce the AnoPas system, a practical implementation from group signatures with time-bound keys. Our contributions are outlined as follows:

(1) We introduce an innovative GS-TBK scheme equipped with essential security features encompassing anonymity, traceability, and non-frameability. In our group signature framework, the group manager establishes distinct time periods to delineate access control prerequisites, associating a signing key with each element in this set to furnish meticulous access control. Distinguishing itself from extant schemes [6–9], our approach employs a tag-based signature [10] to enhance the efficiency of the group signature signing protocol. Concretely, users receive a singular tag-based signature as their signing key for each time period, with all such keys being linked to the user's private key. Consequently, when signing a group signature within the current time frame, users only need to randomize the tag and the tag-based signature corresponding to that time period while simultaneously proving possession of their private key. Our scheme accommodates signature revocation and identity tracing, thereby fortifying security control.

(2) We construct an efficient AnoPas system with our GS-TBK scheme. To deploy AnoPas on mobile phones with smart cards, we allow splitting the user's overall computations securely and efficiently between a smart card and a mobile phone. The smart card holds all secret information used in the AnoPas system and computes a signature of knowledge of the user's private key. The mobile phone randomizes the user's public key and a tag-based signature for the current time period, and the passing protocol can only be executed with the help of the smart card.

(3) We have implemented the AnoPas system using the MIRACL library [11] at an AES-100 bit security level on a PC, specifically, the HUAWEI Matebook 14, equipped with an AMD Ryzen-5 4600H CPU. Our passing protocol achieves a remarkable execution time of 3.8 ms, which is more than 400% faster than the current state-of-the-art solution [9]. Additionally, we have extended our implementation to mobile devices with smart cards, leveraging Aisinochip's smart card chip (ACH512) and a HUAWEI mobile phone (Honor 9i) as the user components. The authority and transport service is provided by a HUAWEI PC (Matebook 14). In a configuration where the smart card's clock frequency is set to 40 MHz, our passing protocol exhibits an efficient performance, requiring approximately 38.6 ms on the smart card and about 33.6 ms on the mobile phone.

1.2. Related work and comparison

1.2.1. Group signatures with time-bound keys

Any GS-TBK scheme implies an anonymous transit pass system, so we will first give the most recent research related to GS-TBK [6–9].

Chu et al. [6,7] introduced a novel variant of group signatures termed "group signatures with time-bound keys". This innovation associates each signing key with an expiration time, thereby possessing properties of both automatic revocation and premature revocation. Their approach melds the short signature scheme pioneered by Boneh et al. [12] with Lin et al.'s 0/1 encoding framework [13]. Notably, the efficiency of their scheme is contingent on the bit-length representation of time. Consequently, the signing cost and group signature size exhibit linearity in $\log n$, with n representing the size of the time period set. However, a limitation of their scheme is that the definition of traceability falls short in capturing the unforgeability of signing key expiration times. This means that any adversary in possession of a signing key associated with an expiration time can generate a valid signature even after the specified expiration time has elapsed.

Subsequent to Chu et al.'s work [6], Emura et al. [8] introduced an enhanced GS-TBK scheme, drawing inspiration from the BBS+ signature scheme [14], Ohara et al.'s revocable group signature scheme [15], and Nakanishiet et al.'s verifier-local revocation group signature scheme [16]. Their contributions include refining the security model of GS-TBK to encompass the unforgeability of signing key expiration times, achieving constant signature cost and size for the first time. Regrettably, the signing protocol of their scheme necessitates 11 bilinear map operations, rendering it impractical for implementation on smart cards.

In [9], Sanders introduced a new approach to constructing a GS-TBK scheme using the unlinkable redactable signature. Unlike previous works, this approach allows the group manager to associate the group signing key with a set of time periods, not just expired times. Compared with [8], Sanders' scheme further reduces the computation cost of signing protocol, which requires only 6 exponentiations in $\mathbb{G}_1$, 2 exponentiations in $\mathbb{G}_2$ and 1 bilinear map operation. Unfortunately, this scheme is still not efficient enough to be implemented on smart cards because it involves 2 exponentiations in $\mathbb{G}_2$, which are more time-consuming than exponentiations in $\mathbb{G}_1$, as well as 1 bilinear operation.

In Table 1, our GS-TBK scheme (Section 3.3.2) is compared with the existing GS-TBK schemes in terms of computational complexities, where we denote the exponentiation in $\mathbb{G}_i$ by e_i ($i \in \{1,2\}$) and denote the bilinear map in pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ by e_p. We also denote by n the size of the set of time periods and by R the size of the revocation list. We see that our scheme improves on existing GS-TBK schemes, and in particular, we significantly reduce the cost of the signing and verification protocols. Our signing protocol requires only 6 exponentiations in $\mathbb{G}_1$, and our verification protocol requires 4 exponentiations in $\mathbb{G}_1$ and 4 bilinear map operations.

Table 1
Computational complexities comparison of group signature schemes with time-bound keys.

Scheme	Group key generation	Join	Issue	Sign	Signature verify	Revocation verify
[6]	$2e_2$	$2e_1 + 2\log n e_2 + 2\log n e_p$	$\log n e_1$	$(3\log n + 3)e_1 + \log n e_2 + (7\log n - 2)e_p$	$3\log n e_1 + \log n e_2 + 7\log n e_p$	$R(1e_1)$
[8]	$1e_1 + 2e_2$	$4e_1 + \log n e_2 + 2\log n e_p$	$(3\log n + 4)e_1$	$23e_1 + 1e_2 + 11e_p$	$24e_1 + 18e_p$	$R(2e_p)$
[9]	$(2n + 1)e_1 + (n + 1)e_2$	$2e_1 + (n + 2)e_2 + 2e_p$	$6e_1 + 2e_2$	$6e_1 + 2e_2 + 1e_p$	$3e_1 + 5e_p$	$R(3e_p)$
Our GS-TBK	$(3 + 2n)e_2$	$4e_1 + 2e_2 + 4ne_p$	$(4 + 3n)e_1 + 2e_2$	$6e_1$	$4e_1 + 4e_p$	$R(3e_p)$

Table 2
Storage complexities comparison of group signature schemes with time-bound keys.

Scheme	Group public key size	Signing key size	Signature size	Revocation list size
[6]	$3\|\mathbb{G}_1\| + 3\|\mathbb{G}_2\|$	$(\log n + 2)\|\mathbb{Z}_p\| + \log n\|\mathbb{G}_1\|$	$1\|\mathbb{Z}_p\| + (5\log n + 2)\|\mathbb{G}_1\| + \log n\|\mathbb{G}_T\|$	$R(2\|\mathbb{Z}_p\|)$
[8]	$8\|\mathbb{G}_1\| + 3\|\mathbb{G}_2\|$	$(2\log n + 1)\|\mathbb{Z}_p\| + \log n\|\mathbb{G}_1\|$	$11\|\mathbb{Z}_p\| + 6\|\mathbb{G}_1\| + 1\|\mathbb{G}_2\|$	$R(2\|\mathbb{G}_1\| + 1\|\mathbb{G}_2\|)$
[9]	$(2n + 1)\|\mathbb{G}_1\| + (n + 1)\|\mathbb{G}_2\|$	$1\|\mathbb{Z}_p\| + 3\|\mathbb{G}_1\|$	$2\|\mathbb{Z}_p\| + 3\|\mathbb{G}_1\| + 1\|\mathbb{G}_2\|$	$R(\|\mathbb{G}_2\|)$
Our GS-TBK	$1\|\mathbb{G}_1\| + (2n + 3)\|\mathbb{G}_2\|$	$1\|\mathbb{Z}_p\| + (3 + n)\|\mathbb{G}_1\|$	$2\|\mathbb{Z}_p\| + 4\|\mathbb{G}_1\|$	$R(\|\mathbb{G}_2\|)$

Table 3
Comparison of AnoPas with analogical transit pass constructions.

Scheme	Computation overheads				Storage overheads	
	Pass/Show (Smart Card)	Pass/Show (Mobile Phone)	Verify	Interactions times	Passing-key/ Credential size	Pass/Token size
[3]	$1e_1$	$3e_1$	$3e_1 + 4e_p + e_T$	4	$2\|\mathbb{Z}_p\| + 1\|\mathbb{G}_1\|$	$5\|\mathbb{Z}_p\| + 1\|\mathbb{G}_1\| + 1\|\mathbb{G}_T\|$
[18]	$5e_1$	$4e_1$	$4e_1 + 4e_p$	1	$1\|\mathbb{Z}_p\| + 4\|\mathbb{G}_1\|$	$2\|\mathbb{Z}_p\| + 4\|\mathbb{G}_1\|$
[19]	$3e_1$	$(n + 2)e_1 + 2e_p$	$(n + 6)e_1 + 2e_2 + (n + 5)e_p$	1	$3\|\mathbb{Z}_p\| + 1\|\mathbb{G}_1\|$	$6\|\mathbb{Z}_p\| + 2\|\mathbb{G}_1\|$
[20]	$3e_1$	$(n + 8)e_1$	$(n + 1)e_1 + 2e_p$	1	$3\|\mathbb{Z}_p\| + 2\|\mathbb{G}_1\|$	$6\|\mathbb{Z}_p\| + 2\|\mathbb{G}_1\|$
[21]	$2e_1$	$7e_1 + 3e_2$	$11e_p$	1	$1\|\mathbb{Z}_p\| + 3n\|\mathbb{G}_1\| + n\|\mathbb{G}_2\|$	$7\|\mathbb{G}_1\| + 2\|\mathbb{G}_2\|$
Our AnoPas	$2e_1$	$4e_1$	$4e_1 + 4e_p$	1	$1\|\mathbb{Z}_p\| + (3 + n)\|\mathbb{G}_1\|$	$2\|\mathbb{Z}_p\| + 4\|\mathbb{G}_1\|$

In Table 2, our GS-TBK scheme (Section 3.3.2) is compared with the existing GS-TBK schemes in terms of storage complexities, where $|\mathbb{Z}_p|, |\mathbb{G}_1|, |\mathbb{G}_2|, |\mathbb{G}_T|$ are the element's sizes in the group $\mathbb{Z}_p, \mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$, respectively. Our scheme achieves minimal signature size as our signature requires only 2 elements in $\mathbb{Z}_p$ and 4 elements in $\mathbb{G}_1$. Our signing key and group public key are larger than those in other works; nonetheless, the signing key and group public key in our scheme can still be stored in a smart card when n is not particularly large. For example, if we use the BN-256 curve [17] and MIRACL [11] to implement our scheme at AES-100 bit security level, each element in $\mathbb{Z}_p, \mathbb{G}_1$, and $\mathbb{G}_2$ takes 40 bytes, 128 bytes, and 272 bytes, respectively. When $n = 100$, which is enough for GS-TBK applications, the group public key and each signing key in our scheme require 54 KB and 13 KB of storage space, respectively. We note that Aisinochip's smart card (ACH512) has 512 KB of Flash, which is sufficient for implementing our scheme.

1.2.2. Transit pass on mobile phone with smart card

Recently proposed cryptographic primitives such as DAA [3,18–20] and anonymous credentials [21], which are specifically implemented for mobile phones with smart cards, can also be used to construct anonymous transit pass systems if their attribute sets are considered as time sets.

Direct Anonymous Attestation (DAA), as applied in practical scenarios, primarily targets implementation on mobile phones with smart cards. In this context, the smart card serves as the Trusted Platform Module (TPM), an internationally recognized standard for security chips. Camenisch et al. established the security model for DAA and introduced multiple schemes [18–20] designed to enhance security and efficiency continually. Core-Helper Anonymous Credentials (CHAC), a concept formalized by Hanzlik and Slamanig [21], consider a constrained core device (e.g., a smart card) and a powerful helper device (e.g., a mobile phone) within their core-helper framework. The fundamental idea revolves around the core device executing operations independently of credential size or attribute count, while the helper device cannot utilize the credential without assistance.

In Table 3, we compare our AnoPas system to the recent DAA and CHAC schemes. The scheme in [3] requires the user and the verifier to interact four times to complete the passing protocol, which is unacceptable in the anonymous transit pass system because the communication delay may lead to service instability. For the other options, we see that our system improves over DAA and CHAC; concretely, in our scheme, the overhead of the mobile phone and the verification algorithm is the same as the current optimal DAA scheme [18], and the overhead on the smart card is the same as the current optimal CHAC scheme [21], i.e., 2 exponentiations in $\mathbb{G}_1$. While our passing keys are larger than other DAA schemes, they are stored on the mobile phone where storage space is more than enough to hold passing keys.

1.3. Organization

The remainder of this paper is structured as follows: Section 2 covers Preliminaries, Section 3 details the construction of our GS-TBK scheme, Section 4 introduces our AnoPas system, Section 5 provides a performance analysis of our AnoPas system, and Section 6 concludes the paper.

2. Preliminary

2.1. Bilinear pairing

Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ represent cyclic groups of prime order p, with g and $\tilde{g}$ serving as generators for $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. The mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ qualifies as a bilinear map when it adheres to three critical properties:

(1) *Bilinearity*: For all $g \in \mathbb{G}_1$, $\widetilde{g} \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}p$, it holds that $e(g^a, \widetilde{g}^b) = e(g, \widetilde{g})^{ab}$.

(2) *Non-Degeneracy*: $e(g, \widetilde{g}) \neq 1_{\mathbb{G}_T}$.

(3) *Computability*: The mapping e must be efficiently computable.

Our scheme is founded upon the Type-III pairing [22], signifying the absence of an efficiently computable homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$.

2.2. Discrete Logarithm (DL) assumption

Let $\mathbb{G}$ denote a cyclic group of prime order with g as its generator. For a pair $(g, g^x) \in \mathbb{G}^2$, the Discrete Logarithm (DL) assumption is satisfied within $\mathbb{G}$ if no efficient adversary can efficiently compute x with non-negligible probability.

2.3. Square discrete logarithm (SDL) assumption

Let $\mathbb{G}$ denote a cyclic group of prime order with g as its generator. When provided with $(g, g^x, g^{x^2}) \in \mathbb{G}^3$, the Square Discrete Logarithm (SDL) assumption is satisfied within $\mathbb{G}$ if no efficient adversary can efficiently compute x with non-negligible probability.

2.4. Decisional square Diffie–Hellman (DSqDH) assumption

Assume $\mathbb{G}$ represents a cyclic group of prime order p with g as its generator. When presented with $(g, g^x, g^y) \in \mathbb{G}^3$, the Decisional Square Diffie–Hellman (DSqDH) assumption is satisfied within the group $\mathbb{G}$ if no efficient adversary can distinguish $y = x^2$ from an element y randomly selected from $\mathbb{Z}_p^*$.

2.5. Signature of knowledge

We employ the concept of Signature of Knowledge (SoK) [23] for an NP-relation $\mathcal{R}$ associated with the language $L_{\mathcal{R}} = y : \exists x, (x, y) \in \mathcal{R}$. This SoK framework comprises a tuple of algorithms, namely (Setup, Sign, Verify), designed to establish the knowledge of private keys.

Setup(1^λ): Given a security parameter λ as input, this algorithm generates and outputs a public parameter denoted as pp.

Sign(m, x, y): With inputs including a message m and a relation $(x, y) \in \mathcal{R}$, this algorithm produces a SoK represented as $\Pi = \mathsf{SoK}\{x : (x, y) \in \mathcal{R}\}$.

Verify(m, Π, y): When provided with a message m, a SoK denoted as Π, and a statement y, this algorithm assesses the validity of Π. If Π is valid, it returns 1; otherwise, it returns 0.

A SoK is deemed $SimExt$ secure [23] if it satisfies the criteria of correctness, simulatability, and extractability. The instantiation of the SoK can be achieved through the application of the Fiat–Shamir paradigm [24], coupled with zero-knowledge protocols outlined in [25].

2.6. Anonymous ephemeral tag

The Anonymous Ephemeral Tag Scheme [10] comprises a tuple of probabilistic polynomial time (PPT) algorithms, denoted as (Setup, GenTag, RandTag, ProveTag, VerifyTag).

Setup(1^λ): This algorithm, upon receiving a security parameter 1^λ, generates a set of public parameters $pp = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \widetilde{g}, p, e)$.

• GenTagpp. Given a public parameter pp, this algorithm randomly selects a private key $usk \xleftarrow{R} \mathbb{Z}_p^*$ and a generator $T_1 \xleftarrow{R} \mathbb{G}_1$, and outputs $T = (T_1, T_2, T_3) \leftarrow (T_1, T_1^{usk}, T_2^{usk})$.

• RandTag(T). Given a tag T, this algorithm selects $r \xleftarrow{R} \mathbb{Z}_p^*$, and outputs $T' = T^r = (T_1', T_2', T_3') \leftarrow (T_1^r, T_2^r, T_3^r)$.

• ProveTag(usk, T). Given the private key usk and a tag T, this algorithm outputs a SoK: $\Pi = \mathsf{SoK}\{usk : T_2 = T_1^{usk}, T_3 = T_2^{usk}\}$.

• VerifyTag(T, Π). Given a tag T and its a proof Π, The verifier outputs 1 if it accepts the proof and 0 otherwise.

2.7. Tag-based signature

The Tag-Based Signature (TBS) scheme, as introduced in [10], is comprised of a tuple of PPT algorithms, denoted as (Setup, KeyGen, Sign, Derive, Verify). • Setup(1^λ). This algorithm, upon receiving a security parameter 1^λ, generates a set of public parameters $pp = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \widetilde{g}, p, e)$. • KeyGen$(n)$. On input an integer n, this algorithm selects $(a, b, c, \{x_i, y_i\}_{i=1}^n) \xleftarrow{R} \mathbb{Z}_p^*$ and computes $\widetilde{A} = \widetilde{g}^a, \widetilde{B} = \widetilde{g}^b, \widetilde{C} = \widetilde{g}^c, \{\widetilde{X}_i = \widetilde{g}^{x_i}, \widetilde{Y}_i = \widetilde{g}^{y_i}\}_{i=1}^n$. The signing key is $sk = (a, b, c, \{x_i, y_i\}_{i=1}^n)$, and the verification key is $vk = (\widetilde{A}, \widetilde{B}, \widetilde{C}, \{\widetilde{X}_i, \widetilde{Y}_i\}_{i=1}^n)$. • Sign$(sk, T, m_i)$. On input a signing key sk, a tag T, and a messages m_i ($m_i \in \mathbb{Z}_p$ and $i \in [1, n]$), this algorithm outputs a signature $\sigma = T_1^{a+x_i+m_i y_i} T_2^b T_3^c$.

• Derive$(vk, T, m_i, \sigma,)$. On input the verification key vk, a tag T, a messages m_i, and a signature σ, this algorithm selects $r \xleftarrow{R} \mathbb{Z}_p^*$, and outputs $T' = T^r, \sigma' = \sigma^r$.

• Verify(vk, T, m_i, σ). A signature σ on m_i and T is valid if the following equation hold: $e(\sigma, \widetilde{g}) = e(T_1, \widetilde{A}\widetilde{X}_i\widetilde{Y}_i^{m_i}) e(T_2, \widetilde{B}) \cdot e(T_3, \widetilde{C})$. If the equation is satisfied, then the algorithm returns 1; otherwise returns 0.

The Tag-Based Signature Scheme [10] exhibits unforgeability in the generic group model [26] when the signing oracle is queried at most once per tag–index pair. Additionally, it achieves unlinkability when the Decisional Square Diffie–Hellman (DSqDH) assumption is computationally hard.

3. Our GS-TBK scheme

In this section, we construct our GS-TBK scheme for a later proposal of the AnoPas system.

3.1. Syntax

We introduce the formal definition of a GS-TBK scheme and establish a crucial distinction: we link a signing key to each element within the active time periods set $\mathcal{T}$, as opposed to solely the overall set of time periods or a single expiration time. This significant distinction empowers users to autonomously request a signing key from the group manager for any specific time period τ within $\mathcal{T}$, provided that τ adheres to the access control criteria defined by the group manager.

Our GS-TBK scheme comprises a set of algorithms, denoted as (GS-TBK.Setup, GS-TBK.GKeyGen, GS-TBK.UKey Gen, GS-TBK.Join, GS-TBK.Sign, GS-TBK.Revoke, GS-TBK. Verify, GS-TBK.Trace), designed to engage three distinct participant roles: the group manager (GM), users (U), and verifiers (V). We present a summary of the notations employed in our GS-TBK scheme, which can be found in Table 4, and proceed to define these algorithms in the subsequent sections formally.

• GS-TBK.Setup$(1^\lambda) \rightarrow pp$. This algorithm inputs a security parameter 1^λ, and outputs the system parameters pp.

• GS-TBK.GKeyGen$(pp, n) \rightarrow (gsk, gpk, \mathcal{REG}, \mathcal{REV})$. This algorithm is operated by GM. It inputs the system parameter pp and a positive integer n and divides the time period for which user access needs to be controlled into n little periods of time, which constitutes a set of time period $[1, n]$. It then generates a group private key gsk and a group public key gpk and initializes a registration list $\mathcal{REG}$ and a revocation list $\mathcal{REV}$. The group public key gpk and the revocation list $\mathcal{REV}$ are publicly, making them known to other participants.

• $\langle \mathsf{GS-TBK.Join}(pp, id, \mathcal{T}) \leftrightarrow GS-TBK.Issue(pp, gsk, gpk, \mathcal{REG}) \rangle \rightarrow (usk, upk, utk, \{\sigma_\tau\}_{\tau \in \mathcal{T}})$. This algorithm is operated by interacting between U and GM, and if $\mathcal{T} \nsubseteq [1, n]$, it returns $\perp$. U takes his identity id and a set of active time periods $\mathcal{T}$ as inputs, and generates a private key usk, a public key upk, and a tracing (and also revocation) key utk. GM takes the group private key gsk, the group public key gpk and the registration list $\mathcal{REG}$ as inputs. After successful completion of this protocol, GM generates a set of signing keys $\{\sigma_\tau\}_{\tau \in \mathcal{T}}$ for U, and stores user's identity id, the set of active time periods $\mathcal{T}$ and the tracing (revocation) key utk to the registration list $\mathcal{REG}$.

Table 4
Notations.

Notation	Description
$1^{\lambda}/\epsilon(\lambda)$	Security parameter/negligible function
$x \xleftarrow{R} \mathbb{Z}_p$	x is randomly selected from the set $\mathbb{Z}_p$
GM/U/V	Group manager/user/verifier
n	The size of the set of time periods
$[1,n]/\mathcal{T}$	The set $\{1,2,\ldots,n\}$/a user's set of active time periods
id	A user's identity
pp	System parameter
gsk/gpk	Group private key/group public key
$\mathcal{REG}/\mathcal{REV}$	Registration list/revocation list
$usk/upk/utk$	Secret/public/tracing (revocation) key of U
τ/σ_τ	Current time period/the signing key with τ
σ	A group signature
R_τ	Revocation information with τ
$\mathcal{H}_1 : \{0,1\}^* \to \mathbb{G}_1$	A collision-resistant hash function
$\mathcal{H}_2 : \{0,1\}^* \to \mathbb{Z}_p$	A collision-resistant hash function
$\perp$	Failure identifier

• GS-TBK.Sign$(pp, \sigma_\tau, usk, upk) \to \sigma$. This algorithm is operated by U. It inputs current time period τ and the corresponding signing key σ_τ, a private key usk and a public key upk, and generates a group signature σ.

• GS-TBK.Revoke$(pp, id, \tau, gsk, \mathcal{REG}, \mathcal{REV}) \to R_\tau$. This algorithm is operated by GM. It inputs a user identity id, a time period τ, the group private key gsk, the registration list $\mathcal{REG}$ and the revocation list $\mathcal{REV}$, and writes a revocation information R_τ to the revocation list $\mathcal{REV}$.

• GS-TBK.Verify$(pp, gpk, \sigma, \mathcal{REV}) \to 0/1$. This algorithm is operated by V. It inputs a group signature σ, the group public key gpk and the revocation list $\mathcal{REV}$, and if σ is valid and not revoked, then it outputs 1, otherwise output 0.

• GS-TBK.Trace$(pp, \sigma, \mathcal{REG}) \to id/\perp$. This algorithm is operated by GM. It inputs a group signature σ and the registration list $\mathcal{REG}$, and outputs either a user identity id or $\perp$.

Correctness. The correctness of our GS-TBK scheme hinges on two essential conditions: (1) a group signature created by an honest user, which has not been revoked by the group manager, should be verifiable by the verifier and traceable by the group manager; (2) a group signature that has been revoked by the group manager must be unverifiable by the verifier. A precise formal definition of correctness is **Supplemental Material A**.

3.2. Security model

A GS-TBK scheme is expected to fulfill specific security requirements, including anonymity, traceability, and non-frameability. Our security model is aligned with the principles outlined in [9], and we offer formal definitions of these security prerequisites. To establish a common framework, we introduce essential global variables and oracles. Within this framework, a designated challenger, denoted as $\mathcal{C}$, bears the responsibility for configuring system parameters and generating keys for the GM. Additionally, $\mathcal{C}$ takes charge of the initialization and oversight of all oracles and global variables. In the context of this security assessment, a probabilistic polynomial-time (PPT) adversary, represented as $\mathcal{A}$, engages with $\mathcal{C}$, who, in turn, emulates the actions of honest participants by utilizing the oracles.

Global Variables:

Within our system, we establish five fundamental sets:

$-\mathcal{HU}$, encompassing the identities of honest users;

$-\mathcal{CU}$, comprising the identities of corrupted users;

$-\mathcal{SK}$, serving as a repository for records $(id, usk, upk, utk, \mathcal{T}, \{\sigma_\tau\}_{\tau\in\mathcal{T}})$ each time the group manager issues a signing key;

$-\mathcal{GS}$, containing entries (id, σ) each time an honest user id generates a group signature σ;

$-\mathcal{TR}$, storing entries (id, σ) whenever the group manager traces a group signature σ.

Oracles:

$-\mathcal{O}_{\text{JoiIss}}(id, \mathcal{T})$. It is an oracle that can be used to play the honest group manager issuing signing keys for an honest user id with the set of time periods $\mathcal{T}$. If $id \in \{\mathcal{CU} \cup \mathcal{HU}\}$ or $\mathcal{T} \subsetneq [1,n]$, it returns $\perp$. Otherwise, the group manager issues the signing keys for the user by running: $\langle \text{GS}-\text{TBK.Join}(pp, id, \mathcal{T}) \leftrightarrow \text{GS}-\text{TBK.Issue}(pp, gsk, gpk, \mathcal{REG})\rangle \to (usk, upk, utk, \{\sigma_\tau\}_{\tau\in\mathcal{T}})$. It adds id to $\mathcal{HU}$ and appends $(id, usk, upk, utk, \mathcal{T}, \{\sigma_\tau\}_{\tau\in\mathcal{T}})$ to $\mathcal{SK}$.

$-\mathcal{O}_{\text{Join}}(id, \mathcal{T})$. It is an oracle that can be used to play the corrupt group manager issuing the signing keys for an honest user id with the set of time periods $\mathcal{T}$. If $id \in \{\mathcal{CU}\cup\mathcal{HU}\}$ or $\mathcal{T} \subsetneq [1,n]$, it returns $\perp$. Otherwise, it obtains the signing keys by running: $\langle \text{GS}-\text{TBK.Join}(pp, id, \mathcal{T}) \leftrightarrow \mathcal{A}(\cdot)\rangle \to (usk, upk, utk, \{\sigma_\tau\}_{\tau\in\mathcal{T}})$, where the group manager's side is executed by the adversary. It adds id to $\mathcal{HU}$ and appends $(id, usk, upk, utk, \mathcal{T}, \{\sigma_\tau\}_{\tau\in\mathcal{T}})$ to $\mathcal{SK}$.

$-\mathcal{O}_{\text{Issue}}(id, \mathcal{T})$. It is an oracle that can be used to play the honest group manager issuing the signing keys for a corrupt user id with the set of time periods $\mathcal{T}$. If $id \in \{\mathcal{CU}\cup\mathcal{HU}\}$ or $\mathcal{T} \subsetneq [1,n]$, it returns $\perp$. Otherwise, it issues the signing keys by running: $\langle \mathcal{A}(\cdot) \leftrightarrow \text{GS}-\text{TBK.Issue}(pp, gsk, gpk, \mathcal{REG})\rangle \to (*, upk, utk, \{\sigma_\tau\}_{\tau\in\mathcal{T}})$, where the user's side is executed by the adversary. It adds id to $\mathcal{CU}$ and appends $(id, *, upk, utk, \mathcal{T}, \{\sigma_\tau\}_{\tau\in\mathcal{T}})$ to $\mathcal{SK}$, where $*$ is the unknown.

$-\mathcal{O}_{\text{CU}}(id)$. It is an oracle that can be used to corrupt an honest user id. If $id \in \mathcal{CU}$, then it returns $\perp$. If $i \in \mathcal{HU}$, then it removes id from $\mathcal{HU}$ and adds id to $\mathcal{CU}$. Finally, it searches for the id from the set $\mathcal{SK}$ and returns $(id, usk, upk, utk, \mathcal{T}, \{\sigma_\tau\}_{\tau\in\mathcal{T}})$.

$-\mathcal{O}_{\text{Sign}}(id, \tau)$. It is an oracle that can be used to play a malicious verifier verifying a group signature for an honest user id. If $id \notin \mathcal{HU}$, then it returns $\perp$. Otherwise, it runs $\langle \text{GS}-\text{TBK.Sign}(pp, \sigma_\tau, usk, upk, \tau) \leftrightarrow \mathcal{A}(\cdot)\rangle \to (\sigma, 0/1)$, where the verifier's side is executed by adversary. Finally, it adds (id, σ) to $\mathcal{GS}$.

$-\mathcal{O}_{\text{Trace}}(\sigma)$. It is an oracle that can be used to play the honest group manager tracing an honest user's identity. It runs the tracing algorithm: $\text{GS}-\text{TBK.Trace}(pp, \sigma, \mathcal{REG}) \to (id/\perp)$. If the trace fails, then it returns $\perp$, else it appends (id, σ) to $\mathcal{TR}$.

$-\mathcal{O}_{\text{Revoke}}(id, \tau)$. It is an oracle that can be used to play the honest group manager revoking a signing key of honest user id with time period τ by running the revocation algorithm: $\text{GS}-\text{TBK.Revoke}(pp, id, \tau, gsk, \mathcal{REV}) \to R_\tau$.

$-\mathcal{O}_{\text{AnCh}_b}(id_0, id_1, \tau)$. It is an oracle that takes as inputs the identity of two honest users who have the same active time period τ. It runs: $\text{GS}-\text{TBK.Sign}(pp, \sigma_{b,\tau}, usk_b, upk_b, \tau) \to \sigma_b$, and returns σ_b. It is a challenge oracle in the anonymity experiment where the adversary must distinguish group signatures of two honest users.

Anonymity.

Anonymity serves as a safeguard for a user's privacy as long as the user's private key remains confidential. Under this circumstance, no verifier can establish a connection between two group signatures generated by the same user when executing the verification algorithm. Importantly, our scheme permits the tracing of a specific group signature without compromising the anonymity of the user's other group signatures.

Definition 1. Anonymity is formalized through the Exp^{anon-b} game, as illustrated in Fig. 1. Our GS-TBK scheme achieves anonymity if, for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, there exists a negligible function $\epsilon(\lambda)$ such that:

$$Adv^{anon} = |\Pr\left[Exp^{anon-1}(\mathcal{A}, \lambda) = 1\right] - \frac{1}{2}| \leqslant \epsilon(\lambda)$$

Traceability. Traceability guarantees that, for any legitimate group signature, the tracing algorithm will not yield $\perp$. In the context of traceable security games, an adversary possesses the capability to access both the registration and revocation lists; however, their objective remains unattainable: they cannot produce a valid group signature that remains untraceable.

$Exp^{anon-b}(\mathcal{A}, \lambda)$ – Anonymity Security Game

1. $\mathcal{HU}, \mathcal{CU}, \mathcal{SK}, \mathcal{GS}, \mathcal{TR} \leftarrow \emptyset$; $pp \leftarrow$ GS-TBK.Setup(1^λ);
2. $(gsk, gpk, \mathcal{REG}, \mathcal{REV}) \leftarrow$ GS-TBK.GKeyGen(pp, n);
3. $\mathcal{O} \leftarrow \{O_{\text{JoinIss}}, O_{\text{CU}}, O_{\text{Sign}}, O_{\text{Trace}}, O_{\text{Revoke}}, O_{\text{AnCh}_b}\}$
4. $b^* \leftarrow \mathcal{A}^{\mathcal{O}}(pp, gpk, \mathcal{REV})$;
5. If $\mathcal{TR}$ stores the outputs of O_{AnCh_b}, then return 0;
6. If $\mathcal{CU}$ or $\mathcal{REV}$ stores an input of O_{AnCh_b}, then return 0;
7. If $b = b^*$, return 1, otherwise, return 0.

$Exp^{trace}(\mathcal{A}, \lambda)$ – Traceability Security Game

1. $\mathcal{HU}, \mathcal{CU}, \mathcal{SK}, \mathcal{GS}, \mathcal{TR} \leftarrow \emptyset$; $pp \leftarrow$ GS-TBK.Setup(1^λ);
2. $(gsk, gpk, \mathcal{REG}, \mathcal{REV}) \leftarrow$ GS-TBK.GKeyGen(pp, n);
3. $\mathcal{O} \leftarrow \{O_{\text{JoinIss}}, O_{\text{Issue}}, O_{\text{CU}}, O_{\text{Sign}}, O_{\text{Trace}}, O_{\text{Revoke}}\}$
4. $\sigma^* \leftarrow \mathcal{A}^{\mathcal{O}}(pp, gpk, \mathcal{REG}, \mathcal{REV})$;
5. If $\sigma^* \in \mathcal{TR}$, then return 0;
6. If $0 \leftarrow$ GS-TBK.Verify$(pp, gpk, \sigma^*, \mathcal{REV})$, then return 0;
7. If $id^* \leftarrow$ GS-TBK.Trace$(pp, \sigma^*, \mathcal{REG})$ and $id^* \in \mathcal{CU}$, then return 0;
8. Return 1.

$Exp^{non-frame}(\mathcal{A}, \lambda)$ - Non-frameability Security Game

1. $\mathcal{HU}, \mathcal{CU}, \mathcal{SK}, \mathcal{GS}, \mathcal{TR} \leftarrow \emptyset$; $pp \leftarrow$ GS-TBK.Setup(1^λ);
2. $(gsk, gpk, \mathcal{REG}, \mathcal{REV}) \leftarrow$ GS-TBK.GKeyGen(pp, n);
3. $\mathcal{O} \leftarrow \{O_{\text{Join}}, O_{\text{CU}}, O_{\text{Sign}}, O_{\text{Trace}}, O_{\text{Revoke}}\}$
4. $\sigma^* \leftarrow \mathcal{A}^{\mathcal{O}}(pp, gsk, \mathcal{REG}, \mathcal{REV})$;
5. If $\sigma^* \in \mathcal{GS}$, then return 0;
6. If $\sigma^* \in \mathcal{TR}$, then return 0;
7. If $0 \leftarrow$ GS-TBK.Verify$(pp, gpk, \sigma^*, \mathcal{REG})$, then return 0;
8. $id^* \leftarrow$ GS-TBK.Trace$(pp, \sigma^*, \mathcal{REG})$ and $id^* \in \mathcal{HU}$, then return 1.
9. Return 0.

Fig. 1. Security games for our GS-TBK scheme.

Definition 2. Traceability is formalized through the Exp^{trace} game, as illustrated in Fig. 1. Our GS-TBK scheme achieves traceability if, for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, there exists a negligible function $\epsilon(\lambda)$ such that:

$$Adv^{trace} = |\Pr\left[Exp^{trace}(\mathcal{A}, \lambda)\right] = 1| \leqslant \epsilon(\lambda)$$

Non-frameability. Non-frameability guarantees that, even in the presence of a malicious group manager, it is impossible to make a false claim that an honest user produced a valid group signature when, in reality, they did not.

Definition 3. The concept of non-frameability is formally defined in game $Exp^{non-frame}$, as illustrated in Fig. 1. Our GS-TBK scheme exhibits framing-resistance when, if for any PPT adversary $\mathcal{A}$, there exists a negligible function $\epsilon(\lambda)$ such that:

$$Adv^{non-frame} = |\Pr\left[Exp^{non-frame}(\mathcal{A}, \lambda)\right] = 1| \leqslant \epsilon(\lambda)$$

3.3. Our construction

3.3.1. Intuition

In our scheme, the anonymous ephemeral tag [10] is used as the users' public key upk, and the tag-based signature [10] is used as the signing key of the group signature. The key idea behind our construction is that any user id joining the group for an active time period τ obtains a tag-base signature σ_τ on a tag–index pair (upk, τ). This means that for each active time period τ in $\mathcal{T}$, the user has an independent signing key σ_τ. We note that the user's private key usk corresponding to all signing keys $\{\sigma_\tau\}_{\tau \in \mathcal{T}}$ of the user id is the same. When issuing a group signature at a time period τ, a user only needs to run Derive algorithm on the tag–signature pair (upk, σ_τ) to get a randomized tag–signature pair (upk', σ'_τ).

To achieve traceability and revocability, a user id has to compute a tracing and revocation key $utk = \widetilde{g}^{usk}$. To achieve non-frameability while allowing revocation, we follow the way as [9] that when the group manager issues the signing keys, it sets the signature message $m_i = usk$ for any index $\tau \in \mathcal{T}$. This method allows us to revoke a tag–index pair (upk, τ) of the user id (the access right for the active time period τ) by testing whether the randomized signature is valid on usk.

3.3.2. Concrete construction

We now introduce our GS-TBK scheme, fulfilling the requirements of Section 3.2.

GS-TBK.Setup$(1^\lambda) \rightarrow pp$: Takes a security parameter 1^λ as input, and outputs a Type-III bilinear pair parameter $pp = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \widetilde{g}, p, e)$.

GS-TBK.GKeyGen$(pp, n) \rightarrow (gsk, gpk, \mathcal{REG}, \mathcal{REV})$: The group manager chooses $(a, b, c, \{x_i, y_i\}_{i=1}^n) \xleftarrow{R} \mathbb{Z}_p^*$, and computes $\widetilde{A} = \widetilde{g}^a$, $\widetilde{B} = \widetilde{g}^b$, $\widetilde{C} = \widetilde{g}^c$, $\{\widetilde{X}_i = \widetilde{g}^{x_i}, \widetilde{Y}_i = \widetilde{g}^{y_i}\}_{i=1}^n$. It then sets $gsk = (a, b, c, \{x_i, y_i\}_{i=1}^n)$ and $gpk = (\widetilde{A}, \widetilde{B}, \widetilde{C}, \{\widetilde{X}_i, \widetilde{Y}_i\}_{i=1}^n)$, and initializes a registration list $\mathcal{REG} \leftarrow \emptyset$ and a revocation list $\mathcal{REV} \leftarrow \emptyset$.

$\langle$GS – TBK.Join$(pp, id, \mathcal{T}) \leftrightarrow$ GS – TBK.Issue$(pp, gsk, gpk, \mathcal{REG})\rangle \rightarrow (usk, upk, utk, \{\sigma_\tau\}_{\tau \in \mathcal{T}})$: As shown in Fig. 2, U interacts with GM to generate the user's signing keys $\{\sigma_\tau\}_{\tau \in \mathcal{T}}$.

$\langle$GS-TBK.Join$(pp, id, \mathcal{T}) \leftrightarrow$ GS-TBK.Issue$(pp, gsk, gpk, \mathcal{REG})\rangle \rightarrow (usk, upk, utk, \{\sigma_\tau\}_{\tau\in\mathcal{T}})$

User: U		Group Manager: GM
chooses $usk \xleftarrow{R} \mathbb{Z}_p^*$, and computes: $upk = (T_1, T_2, T_3) \leftarrow (\mathcal{H}_1(id), T_1^{usk}, T_2^{usk})$; $utk = \widetilde{g}^{usk}$; $\Pi = \{usk : T_2 = T_1^{usk}, T_3 = T_2^{usk}, utk = \widetilde{g}^{usk}\}$.	$\xrightarrow{upk, utk, \Pi, \mathcal{T}}$	verifies Π. for all $\tau \in \mathcal{T}$, computes $\sigma_\tau = T_1^{a+x_\tau} T_2^{y_\tau+b} T_3^c$.
for all $\tau \in \mathcal{T}$, verifies $e(\sigma_\tau, \widetilde{g}) = e(T_1, \widetilde{AX}_\tau) \cdot e(T_2, \widetilde{BY}_\tau) e(T_3, \widetilde{C})$. stores $(usk, upk, \mathcal{T}, \{\sigma_\tau\}_{\tau\in\mathcal{T}})$.	$\xleftarrow{\{\sigma_\tau\}_{\tau\in\mathcal{T}}}$	stores $\mathcal{REG}[id] \leftarrow (id, \mathcal{T}, upk, utk)$.

Fig. 2. Definition of the GS-TBK.Join and GS-TBK.Issue algorithms.

- A new user id chooses a randomly private key $usk \xleftarrow{R} \mathbb{Z}_p^*$, and computes his public key $upk = (T_1, T_2, T_3) \leftarrow (\mathcal{H}_1(id), \mathcal{H}_1(id)^{usk}, \mathcal{H}_1(id)^{usk^2})$ and tracing (revocation) key $utk = \widetilde{g}^{usk}$. We note that T_1 is a given and randomly chosen generator of $\mathbb{G}_1$ for user id. He then proves the knowledge of usk using SoK [23]: $\Pi = \{usk : T_2 = T_1^{usk}, T_3 = T_2^{usk}, utk = \widetilde{g}^{usk}\}$, and sends $(id, \mathcal{T}, upk, utk, \Pi)$ to the group manager.
- If the SoK Π is valid, the group manager computes $\sigma_\tau = T_1^{a+x_\tau} T_2^{y_\tau+b} T_3^c$ for all $\tau \in \mathcal{T}$ and returns $\{\sigma_\tau\}_{\tau\in\mathcal{T}}$, where σ_τ is a valid tag-based signature on the tag–index pair (upk, τ) and message $m_\tau = usk$. In the meantime, the group manager stores $(id, \mathcal{T}, upk, utk)$ to $\mathcal{REG}[id]$.
- The user is then able to verify the validity of σ_τ for each $\tau \in \mathcal{T}$ by checks equation $e(\sigma_\tau, \widetilde{g}) = e(T_1, \widetilde{AX}_\tau) \cdot e(T_2, \widetilde{BY}_\tau) \cdot e(T_3, \widetilde{C})$, and then stores the signing keys $(usk, \mathcal{T}, \{\sigma_\tau\}_{\tau\in\mathcal{T}})$ and public key upk.

GS-TBK.Sign$(pp, \sigma_\tau, usk, upk) \rightarrow \sigma$: In signing protocol, we want to let the user prove the knowledge of a tag-based signature with the current time period τ.

- Firstly, the user chooses a random $r \xleftarrow{R} \mathbb{Z}_p^*$, and randomizes the public key $upk' = (T_1', T_2', T_3') \leftarrow (T_1^r, T_2^r, T_3^r)$ and signature $\sigma_\tau' = \sigma_\tau^r$.
- Then, he chooses a random $k \xleftarrow{R} \mathbb{Z}_p^*$, and proves the knowledge of private key usk by computing: $R_1 = T_1'^k, R_2 = T_2'^k$; $c = \mathcal{H}_2(T_1', T_2', T_3', R_1, R_2, \tau, nonce)$; $s = k - c \cdot usk$.
- Finally, he sets $\sigma = (\tau, upk', \sigma_\tau', c, s)$, and sends σ to a verifier.

To prevent replay attacks, the *nonce* is randomly generated by the verifier and sent to the user before the user initiates the signing protocol.

GS-TBK.Revoke$(pp, id, \tau, gsk, \mathcal{REG}, \mathcal{REV}) \rightarrow R_\tau$. The group manager computes $R_\tau = \widetilde{g}^{x_\tau} utk^{y_\tau}$, and stores $\mathcal{REV} \leftarrow R_\tau$.

GS-TBK.Verify$(pp, \sigma, gpk, \mathcal{REV}) \rightarrow 0/1$: The verifier runs GS-TBK.Verify algorithm when receives σ from a user.

- Firstly, it runs signature verification: if $c \neq \mathcal{H}_2(T_1', T_2', T_3', T_1'^s T_2'^c, T_2'^s T_3'^c, \tau, nonce)$, then outputs 0. if $e(\sigma_\tau', \widetilde{g}) \neq e(T_1', \widetilde{AX}_\tau) e(T_2', \widetilde{BY}_\tau) e(T_3', \widetilde{C})$, then outputs 0.
- Then, it runs revocation verification: if $\exists R_\tau \in \mathcal{REV}$, $e(T_1', \widetilde{X}_\tau) e(T_2', \widetilde{Y}_\tau) = e(T_1', R_\tau)$, then outputs 0, else outputs 1.

The GS-TBK.Sign and GS-TBK.Verify algorithms are depicted in Fig. 3.

GS-TBK.Trace$(pp, \sigma, \mathcal{REG}) \rightarrow id/\perp$. For each $(id, \tau, upk, utk) \in \mathcal{REG}$, the group manager tests whether $e(T_1', utk) = e(T_2', \widetilde{g})$ and $e(T_2', utk) = e(T_3', \widetilde{g})$, until it gets a match, in which case it returns the corresponding user identity id.

3.4. Security analysis

The correctness of our GS-TKB scheme is analyzed in **Supplemental Material B**. In this section, we define the following theorems to formalize that our construction from Section 3.3.2 satisfies all the desired security guarantees defined in Section 3.2.

3.4.1. Anonymity

In this subsection, we provide proof of anonymity.

Theorem 1. *Our GS-TBK scheme achieves anonymity if DSqDH assumption holds in* $\mathbb{G}_1$.

Proof. Let $\mathcal{A}$ be an adversary against the anonymity of our GS-TBK scheme and $D_{sqdh} = (h, h^x, h^y)$ be a DSqDH challenge in $\mathbb{G}_1$. At the beginning of the anonymity game Exp^{anon-b}, $\mathcal{C}$ runs $pp \leftarrow$ GS − TBK.Setup(1^λ) and $(gsk, gpk, \mathcal{REG}, \mathcal{REV}) \leftarrow$ GS − TBK.GKeyGen (pp, n), and sends (pp, gpk) to $\mathcal{A}$ that eventually returns $(id_0, id_1, \mathcal{T}^*)$. $\mathcal{C}$ guesses the identity of the user id^*. For each tag–index pairs (D_{sqdh}, τ^*), the user id^* will have a tag-based signature σ_τ^*, where $\tau^* \in \mathcal{T}^*$. $\mathcal{C}$ answers the oracles query as follows.

• $\mathcal{O}_{\mathsf{JoiIss}}$: If $id \in \mathcal{CU}$ or $id \in \mathcal{HU}$, then returns $\perp$. If $id \neq id^*$, then $\mathcal{C}$ proceeds as usual. Otherwise, it sets $upk^* = D_{sqdh}$, and issues a valid tag-based signature σ_{τ^*} for each tag–index pairs (D_{sqdh}, τ^*).

• $\mathcal{O}_{\mathsf{Sign}}$: If $id \neq id^*$, then $\mathcal{C}$ proceeds as usual. Otherwise, it simulates the SoK of user's private key.

• $\mathcal{O}_{\mathsf{CU}}, \mathcal{O}_{\mathsf{Trace}}, \mathcal{O}_{\mathsf{Revoke}}$: If $id \neq id^*$, then $\mathcal{C}$ proceeds as usual. Otherwise, it returns $\perp$.

At some state in the game, $\mathcal{A}$ outputs two identities (id_0, id_1), along with a set $\mathcal{T}^*$ of time periods. If $id^* \neq id_b$, then $\mathcal{C}$ aborts. Else, it answers the oracles query as follows.

• $\mathcal{O}_{\mathsf{AnCh}_b}$: It chooses a random $r \xleftarrow{R} \mathbb{Z}_p^*$, and computes $\sigma_{\tau^*}{}' = \sigma_{\tau^*}^r$ and $upk^{*\prime} = upk^{*r}$. Then it simulates the SoK of user's private key.

If $y = x^2$, the simulation is perfect. Else, y is random, which means that $upk^{*\prime} = (h^r, h^{xr}, h^{yr})$ are random elements in $\mathbb{G}_1$, and $\mathcal{A}$ cannot succeed in this game with non-negligible advantage. Therefore, any change in the behavior of $\mathcal{A}$ can be used to solve the DSqDH problem in $\mathbb{G}_1$, unless $\mathcal{C}$ aborts. The advantage of $\mathcal{C}$ is then at least $\frac{\epsilon}{m}$, where m is a bound on the number of honest users.

3.4.2. Traceability

In this subsection, we provide proof of traceability.

Theorem 2. *Our GS-TBK scheme achieves traceability if the tag-based signature is unforgeable.*

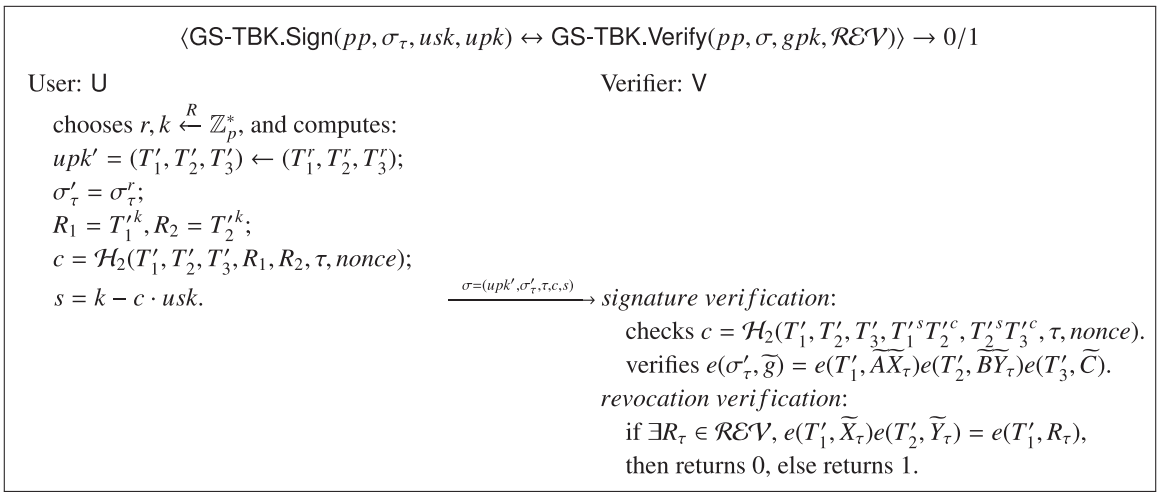

Fig. 3. Definition of the GS-TBK.Sign and GS-TBK.Verify algorithms.

Proof. $\mathcal{C}$ runs the unforgeability game of the tag-based signature and so receives public parameters $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \widetilde{g}, p, e, \widetilde{A}, \widetilde{B}, \widetilde{C}, \{\widetilde{X}_i, \widetilde{Y}_i\}_{i=1}^n)$. $\mathcal{C}$ runs GS-TBK.GKeyGen to generate remaining parameters, and sends (pp, gpk) to $\mathcal{A}$. $\mathcal{C}$ can ask the oracle of the tag-based signature, $\mathcal{O}_{TBSign}(\cdot)$, with an unlimited number of times. $\mathcal{C}$ answers oracles queries as follows:

• $\mathcal{O}_{\text{Issue}}$: $\mathcal{C}$ runs the extractor of Π to recover the private key usk and then submits $(usk, \mathcal{T})$ to the signing oracle $\mathcal{O}_{TBSign}(\cdot)$. It then receives a set of tag-based signatures $\{\sigma_\tau\}_{\tau\in\mathcal{T}}$.

• $\mathcal{O}_{\text{JoiIss}}$: $\mathcal{C}$ can simulate this oracle as well as $\mathcal{O}_{\text{Issue}}$.

• $\mathcal{O}_{\text{Sign}}$: $\mathcal{C}$ knows users' private keys and so perfectly simulates this oracle.

• $\mathcal{O}_{\text{Revoke}}$: $\mathcal{C}$ queries the private key usk of the user id stored in the set SK and computes $R_\tau = \widetilde{X}_\tau \widetilde{Y}_\tau^{usk}$.

• $\mathcal{O}_{\text{Trace}}$: $\mathcal{C}$ knows user's tracing (revocation) key and so perfectly simulates this oracle.

At the end of the game, if $\mathcal{A}$ can prove possession of a signing key σ^* on the tag–index pair (upk^*, τ^*) with probability ϵ and trace failed, then $\mathcal{C}$ extracts usk^* from $\sigma^* = (\tau^*, upk'^*, \sigma'^*_\tau, c^*, s^*)$. Because the trace failed, for any honest user $id \in \mathcal{HU}$, usk^* must be different from private key usk of user id, this means that σ'^*_τ is a valid forgery against the tag-based signature of message usk^* on the tag–index pair (upk^*, τ^*).

3.4.3. Non-frameability

In this subsection, we provide proof of non-frameability.

Theorem 3. *Our GS-TBK scheme achieves non-frameability if SDL assumption holds in $\mathbb{G}_1$ and DL assumption holds in $\mathbb{G}_2$.*

Proof. Let $\mathcal{A}$ be an adversary against the non-frameability of our GS-TBK scheme. Let (h, h^x, h^{x^2}) be a SDL challenge in $\mathbb{G}_1$ and $(\widetilde{g}, \widetilde{g}^x)$ be a DL challenge in $\mathbb{G}_2$. At the beginning of the non-frameability game $Exp^{non-frame}$, $\mathcal{C}$ runs $pp \leftarrow \text{GS} - \text{TBK.Setup}(1^\lambda)$ and $(gsk, gpk, \mathcal{REG}, \mathcal{REV}) \leftarrow \text{GS} - \text{TBK.GKeyGen}\ (pp, n)$ to generate remaining parameters, and sends $(pp, gpk, gsk,)$ to $\mathcal{A}$. $\mathcal{A}$ can issue signing keys, trace user identity, and read or write the registration list. $\mathcal{C}$ guesses an identity of honest user id^*, that $\mathcal{A}$ aims to frame, and implicitly sets the private key of id^* to $usk^* = x$. $\mathcal{C}$ abort if this guess is wrong. $\mathcal{C}$ answers the oracles query as follows.

• $\mathcal{O}_{\text{Join}}$: If $id \neq id^*$, then $\mathcal{C}$ proceeds as usual. Otherwise, it assigns $upk^* = (h, h^x, h^{x^2})$ and $utk^* = \widetilde{g}^x$, and simulates the signature of knowledge of user's private key x.

• $\mathcal{O}_{\text{CU}}$: If $id \neq id^*$, then $\mathcal{C}$ proceeds as usual. Otherwise, it returns $\perp$.

• $\mathcal{O}_{\text{Sign}}$: If $id \neq id^*$, then $\mathcal{C}$ proceeds as usual. Otherwise, it simulates the signature of knowledge of user's private key x.

• $\mathcal{O}_{\text{Trace}}, \mathcal{O}_{\text{Revoke}}$: $\mathcal{C}$ knows the group manager's private keys and so perfectly simulates these oracles.

Eventually, $\mathcal{A}$ returns a valid group signature $\sigma^* = (\tau^*, upk'^*, \sigma'_{\tau^*}, c^*, s^*)$ that can be traced back to id^*. $\mathcal{C}$ can extract x from the signature of knowledge to solve the SDL and DL problems. Any adversary $\mathcal{A}$ succeeding against the non-frameability of our scheme with probability ϵ can then be used to solve the SDL and DL problems with probability $\frac{\epsilon}{m}$, where m is a bound on the number of honest users.

4. Anonymous transit pass system

4.1. Basic design of AnoPas

This section presents the basic design of our AnoPas system from the GS-TBK scheme of Section 3.3. As shown in Fig. 4, the AnoPas system involves three types of participants: authority, users, and transport services, and consists of seven algorithms, which are Setup, Request, Visa, Pass, Revoke, Verify and Trace. The AnoPas system can be trivially obtained using our GS-TBK scheme. The following is a functional description of all algorithms and their construction:

• Setup: The authority executes this algorithm to initialize the system. The specific operation is to run GS-TBK.Setup to generate the public parameters, then run GS-TBK.GKeyGen to generate the master key.

• Request: A user executes this algorithm by running GS-TBK.Join to request the passing keys from the authority.

• Visa: The authority executes this algorithm by running GS-TBK.Issue to issue passing keys to users.

• Pass: A user executes this algorithm by running GS-TBK. Sign to compute a transit pass for access transport services.

• Revoke: The authority executes this algorithm by running GS-TBK.Revoke to revoke user privileges prematurely.

• Verify: A transport service executes this algorithm by running GS-TBK.Verify to verify user permissions.

• Trace: The authority executes this algorithm by running GS-TBK.Trace to recover the identity of the malicious user.

With the widespread use of mobile phones with smart cards (e.g., SIM cards), deploying transit passes on them is an ideal choice in terms of security and portability. In order to deploy our AnoPas system in mobile phones with smart cards, the Request (i.e., GS-TBK.Join) and Pass (i.e., GS-TBK.Sign) algorithms executed by the user in the system should be split into two parts without compromising its security. Among them, the smart card stores the user's private key and performs the key-related part, and the operations for the mobile phone should be

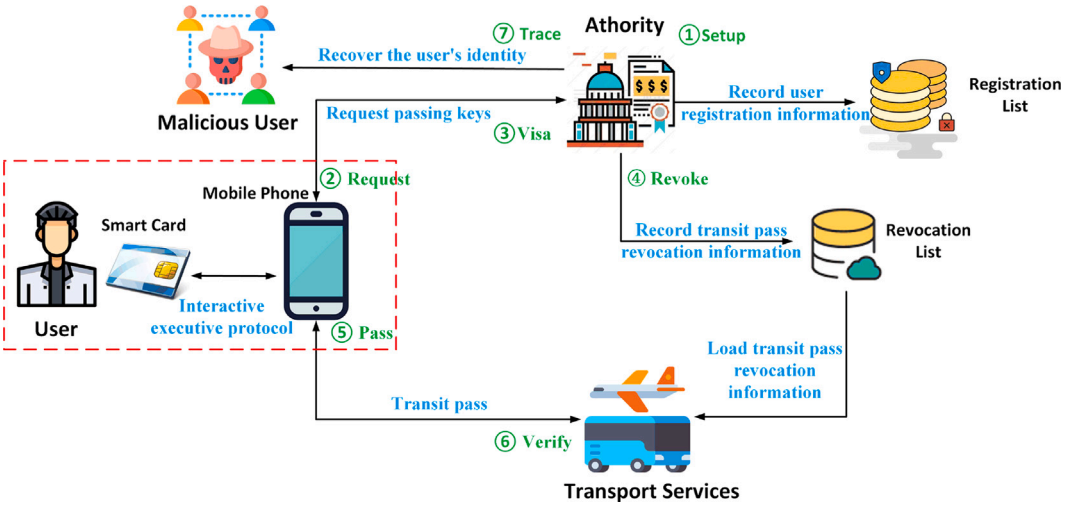

Fig. 4. System architecture of our AnoPas system.

$Exp^{dep}(\mathcal{A}, \lambda)$

1. $\mathcal{HU}, \mathcal{CU}, \mathcal{SK}, \mathcal{GS}, \mathcal{TR} \leftarrow \emptyset$; $pp \leftarrow$ GS-TBK.Setup(1^{λ});
2. $(gsk, gpk, \mathcal{REG}, \mathcal{REV}) \leftarrow$ GS-TBK.GKeyGen(pp, n);
3. $O \leftarrow \{O_{\text{SCJoin}}, O_{\text{Issue}}, O_{\text{CU}}, O_{\text{SCSign}}\}$;
4. $\sigma^* \leftarrow \mathcal{A}^{O}(pp, gpk, \mathcal{REG}, \mathcal{REV})$;
5. If $\sigma^* \in \mathcal{GS}$, then return 0;
6. If $0 \leftarrow$ GS-TBK.Verify$(pp, gpk, \sigma^*, \mathcal{REG})$, then return 0;
7. $id^* \leftarrow$ GS-TBK.Trace$(pp, \sigma^*, \mathcal{REG})$ and $id^* \in \mathcal{HU}$, then return 1.
8. Return 0.

Fig. 5. Dependability security game.

key-independent with a relatively more time-consuming part. The (malicious) mobile phone cannot complete the Request (i.e., GS-TBK.Join) and Pass (i.e., GS-TBK.Sign) algorithms without the help of the smart card.

4.2. Divide user computations of GS-TBK

Our GS-TBK scheme efficiently distributes user computations between a trusted, yet constrained device, exemplified by a smart card in Fig. 4, and a more powerful helper device, such as a mobile phone shown in Fig. 4. Our central concept involves the smart card handling user key generation and the proof of user private key knowledge. Meanwhile, the mobile phone executes operations independent of the user's private key, specifically related to public key randomization and tag-based signature generation. We will now elucidate the technique facilitating the division of the GS-TBK.Join and GS-TBK.Sign algorithms between the smart card and the mobile phone.

The GS-TBK.Join algorithm entails three main steps for the user: generating the user key, computing the signature of knowledge Π, and verifying the tag-based signatures. The smart card handles user key generation and the signature of knowledge since both tasks necessitate knowledge of the user's private key. In contrast, the mobile phone performs the verification of tag-based signatures as it does not require access to the user's private key and can execute bilinear map operations, which are typically unsupported by lightweight devices.

Split GS-TBK.Join algorithm:

- Smart Card:
  (1) User key generation: chooses $usk \xleftarrow{R} \mathbb{Z}_p^*$, and computes: $upk = (T_1, T_2, T_3) \leftarrow (\mathcal{H}_1(id), T_1^{usk}, T_2^{usk})$; $utk = \tilde{g}^{usk}$.
  (2) The SoK of user's private key: $\Pi = \{usk : T_2 = T_1^{usk}, T_3 = T_2^{usk}, utk = \tilde{g}^{usk}\}$.
  (3) Stores usk and sends (upk, utk, Π) to the helper device.
- Mobile Phone:
  (1) Verify the tag-based signatures: for all $\tau \in \mathcal{T}$, verifies $e(\sigma_\tau, \tilde{g}) = e(T_1, \widetilde{AX}_\tau)e(T_2, \widetilde{BY}_\tau)e(T_3, \tilde{C})$.
  (2) Stores $(upk, \mathcal{T}, \{\sigma_\tau\}_{\tau \in \mathcal{T}})$.

The GS-TBK.Sign algorithm requires the user to randomize the user's public key and the tag-based signature, and to compute the signature of knowledge. These operations are efficient because they only require 6 exponentiations in $\mathbb{G}_1$. We can still split the GS-TBK.Sign algorithm into two parts according to the principle of splitting the GS-TBK.Join algorithm, because this will reduce the computation of the smart card to 2 exponentiations in $\mathbb{G}_1$.

Split GS-TBK.Sign algorithm:

- Mobile Phone:
  (1) Randomization of user's public key: chooses $r \xleftarrow{R} \mathbb{Z}_p^*$, and computes $upk' = (T_1', T_2', T_3') \leftarrow (T_1^r, T_2^r, T_3^r)$.
  (2) Randomization of the tag-based signature: computes $\sigma_\tau' = \sigma_\tau^r$.
  (3) Sends $(upk', nonce)$ to the core device.
- Smart Card:
  (1) The SoK of user's private key: chooses $k \xleftarrow{R} \mathbb{Z}_p^*$, and computes $R_1 = T_1'^k, R_2 = T_2'^k$; $c = \mathcal{H}_2(T_1', T_2', T_3', R_1, R_2, \tau, nonce)$; $s = k - c \cdot usk$.
  (2) Returns (c, s) to the helper device.

4.3. Security analysis of AnoPas

The AnoPas system can be constructed directly using the GS-TBK scheme, ensuring anonymity, traceability, and non-frameability (unforgeability) properties based on the security of the GS-TBK scheme. Additionally, to address the constraints of the AnoPas deployment environment, we divide the user computing part of the GS-TBK.Join (i.e., AnoPas.Request) and GS-TBK.Sign (i.e., AnoPas.Pass) algorithms into smart cards and mobile phones, the latter of which may be malicious. A malicious mobile phone could attempt to compute a transit pass without the involvement of the smart card.

Dependability. Dependability ensures that even if an adversary takes control of the mobile phone, he should not be able to compute a group signature (i.e., a transit pass) in a given session in place of an honest user without interacting with the smart card.

To formally define the dependability, we need to add some oracles, as shown below.

$-\mathcal{O}_{\mathsf{SCJoin}}(id, \mathcal{T})$. It is an oracle that can be used to play the honest group manager, issuing the signing keys for an honest user id with a malicious mobile phone. If $id \in \{\mathcal{CU} \cup \mathcal{HU}\}$ or $\mathcal{T} \subsetneq [1, n]$, it returns $\perp$. Otherwise, it obtains the signing keys by running: $\langle\langle \mathsf{GS-TBK.Join.SmartCard}(pp, id) \leftrightarrow \mathcal{A}(pp, id, \mathcal{T}, \cdot)\rangle \leftrightarrow \mathsf{GS-TBK.Issue}(pp, gsk, gpk, \mathcal{REG})\rangle \rightarrow (usk, upk, utk, \{\sigma_\tau\}_{\tau\in\mathcal{T}})$, where the adversary executes the side of the mobile phone. It adds id to $\mathcal{HU}$ and appends $(id, usk, upk, utk, \mathcal{T}, \{\sigma_\tau\}_{\tau\in\mathcal{T}})$ to $\mathcal{SK}$.

$-\mathcal{O}_{\mathsf{SCSign}}(id, \tau)$. It is an oracle that can be used to play an honest verifier verifying a group signature for an honest user id with a malicious mobile phone. If $id \notin \mathcal{HU}$, then it returns $\perp$. Otherwise, it runs $\langle\langle \mathsf{GS-TBK.Sign.SmartCard}(pp, usk, upk) \leftrightarrow \mathcal{A}(pp, \sigma_\tau, \tau, \cdot)\rangle \leftrightarrow \mathsf{GS-TBK.Verify}(pp, gpk, \sigma, \mathcal{REV})\rangle \rightarrow (\sigma, 0/1)$, where the adversary executes the side of the mobile phone. Finally, it adds (id, σ) to $\mathcal{GS}$.

Definition 4. The concept of dependability is formally defined in game Exp^{dep}, as illustrated in Fig. 5. Our GS-TBK scheme achieves dependability, if for any PPT adversary $\mathcal{A}$, there exists a negligible function $\epsilon(\lambda)$ such that:

$$Adv^{dep} = |\Pr\left[Exp^{dep}(\mathcal{A}, \lambda)\right] = 1| \leqslant \epsilon(\lambda)$$

Next, we provide proof of dependability.

Theorem 4. *The dependability of our AnoPas system is achieved if SDL assumption holds in $\mathbb{G}_1$ and DL assumption holds in $\mathbb{G}_2$.*

Proof. Let $\mathcal{A}$ be an adversary against the dependability of our GS-TBK scheme. Let (h, h^x, h^{x^2}) be a SDL challenge in $\mathbb{G}_1$ and $(\widetilde{g}, \widetilde{g}^x)$ be a DL challenge in $\mathbb{G}_2$. At the beginning of the dependability game Exp^{dep}, $\mathcal{C}$ runs $pp \leftarrow \mathsf{GS-TBK.Setup}(1^\lambda)$ and $(gsk, gpk, \mathcal{REG}, \mathcal{REV}) \leftarrow \mathsf{GS-TBK.GKeyGen}(pp, n)$ to generate remaining parameters, and sends (pp, gpk) to $\mathcal{A}$. $\mathcal{C}$ guesses an identity of honest user id^*, that $\mathcal{A}$ aims to forge, and implicitly sets the private key of id^* to $usk^* = x$. $\mathcal{C}$ abort if this guess is wrong. $\mathcal{C}$ answers the oracles query as follows.

- $\mathcal{O}_{\mathsf{SCJoin}}$: If $id \neq id^*$, then $\mathcal{C}$ proceeds as usual. Otherwise, it assigns $upk^* = (h, h^x, h^{x^2})$ and $utk^* = \widetilde{g}^x$, and simulates the signature of knowledge of user's private key x.
- $\mathcal{O}_{\mathsf{Issue}}$: $\mathcal{C}$ knows the group manager' private key and so perfectly simulates this oracle.
- $\mathcal{O}_{\mathsf{CU}}$: If $id \neq id^*$, then $\mathcal{C}$ proceeds as usual. Otherwise, it returns $\perp$.
- $\mathcal{O}_{\mathsf{SCSign}}$: If $id \neq id^*$, then $\mathcal{C}$ proceeds as usual. Otherwise, it simulates the signature of knowledge of user's private key x.

Eventually, $\mathcal{A}$ returns a valid group signature $\sigma^* = (\tau^*, upk'^*, \sigma'_{\tau^*}, c^*, s^*)$ that can be traced back to id^*. $\mathcal{C}$ can extract x from the signature of knowledge to solve the SDL and DL problems. Any adversary $\mathcal{A}$ succeeding against the dependability of our scheme with probability ϵ can then be used to solve the SDL and DL problems with probability $\frac{\epsilon}{m}$, where m is a bound on the number of honest users.

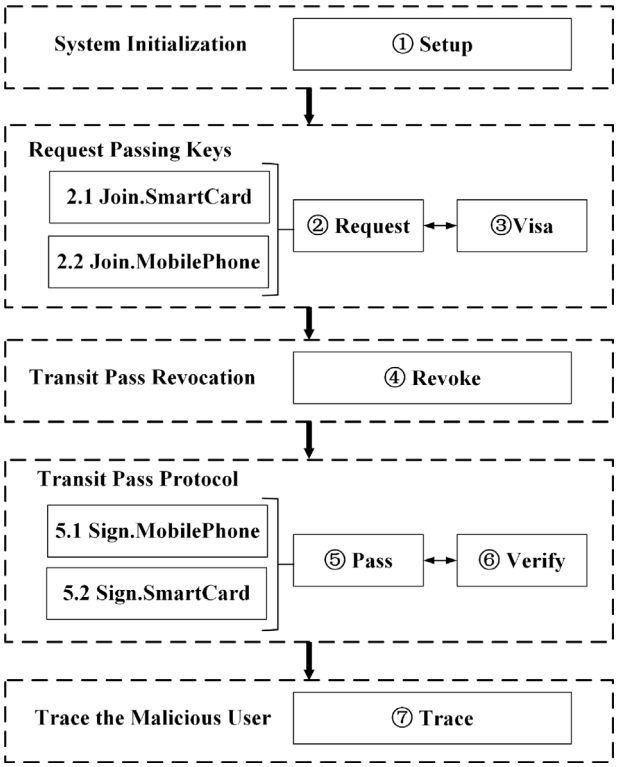

Fig. 6. Workflow of our AnoPas system.

4.4. AnoPas on mobile phone with smart card

Following the division of user computations in the GS-TBK scheme, the operational workflow of our AnoPas system on a mobile phone with a smart card is delineated below.

As shown in Figs. 4 and 6, the authority sets up the public parameters and generates the system's master key (step ①). When a user requests the passing keys from the authority (step ②), his smart card first selects a private key and securely stores it. Then, the smart card calculates the user's public key, the tracing (revocation) key and the proof of knowledge for the user's private key (step 2.1) and transmits this data to the mobile phone. The mobile phone initiates a request for the passing keys (step 2.2), and the authority then issues these keys to the user while recording the user's registration information (step ③). After the visa protocol is completed, the passing keys of multiple time periods are verified and stored in the mobile phone. In case of premature revocation, the authority logs the revocation details in the revocation list (step ④). For a transit passing protocol (step ⑤), the mobile phone first acquires a nonce generated by the transport service and initiates a request (step 5.1), which is then transmitted to the smart card. The smart card computes the SoK for the user's private key and returns the results (step 5.2). Finally, the mobile phone forwards the resulting transit pass to the transport service, which downloads the list of revocation information and verifies the validity and correctness of the transit pass (step ⑥). If it is necessary to identify a transit pass generated by a malicious user, the authority can recover the user's identity (step ⑦).

5. Performance analysis

In this section, we assess the performance of our AnoPas system. To provide a precise evaluation of its efficiency, we showcase its implementations on PC, smart card, and mobile phones with smart cards, respectively. Furthermore, we perform a comparative analysis

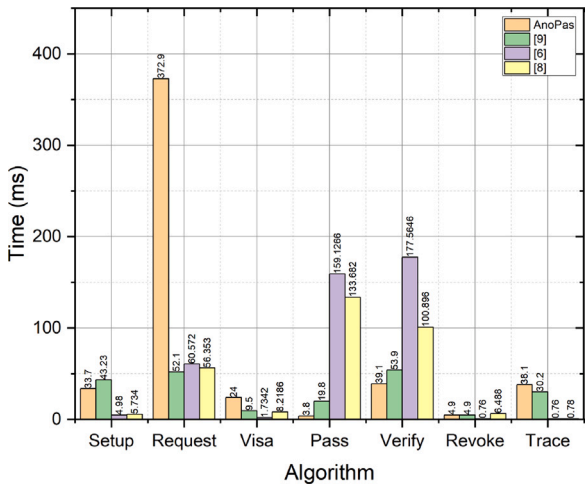

Fig. 7. Execution time on a PC.

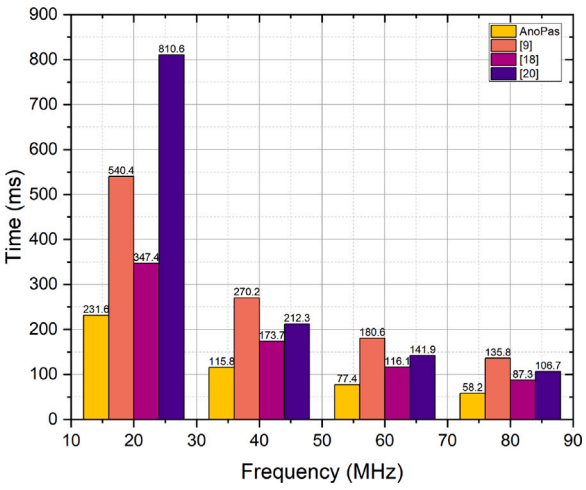

Fig. 9. Execution time of Pass algorithm on a smart card.

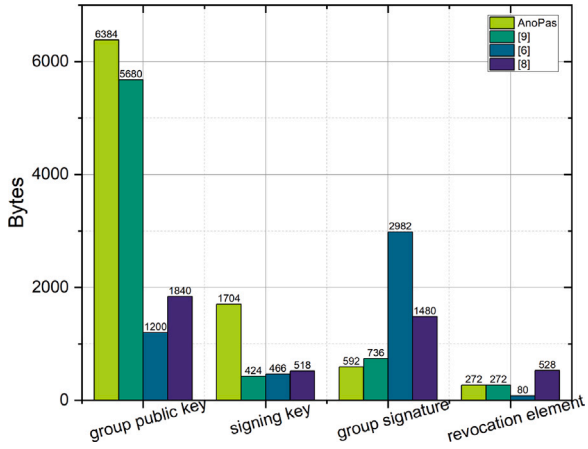

Fig. 8. Storage and communication cost.

against the transit pass systems [6,8,9] and analogous transit pass systems [18–21] suitable for deployment on the mobile phone with smart card. Our implementations employ the Type-III pairing [22] and the Barreto–Naehrig curve (BN-256) [17] at the AES-100 bit security level.

5.1. Implementation on a PC

We have implemented our AnoPas system and evaluated its performance on a HUAWEI Matebook 14 laptop equipped with an AMD Ryzen-5 4600H CPU with a clock speed of 3.00 GHz. This laptop configuration includes 16 GB of RAM and a 512 GB SSD, running the Ubuntu Kylin 16.04 operating system. Our experiments were performed using the MIRACL library [11].

In Fig. 7, we compare the computation cost of all algorithms in our system with the most efficient transit pass [6,8,9] at $n = 10$. The initial algorithm Setup is executed by *authority*, the passing keys request algorithm Request by *user*, the passing keys issuing algorithm Visa by *authority*, the pass algorithm Pass by *user*, the verification algorithm Verify by *transport service*, and the revocation algorithm Revoke and tracing algorithm Trace by *authority*. In our scheme, the computation costs of Setup, Request, Visa, Pass, Verify, Revoke and Trace are 33.7 ms, 372.9 ms, 24 ms, 3.8 ms, 39.1 ms, 4.9 ms, and 38.1 ms, respectively. In the systems of [6,8,9], the computation costs of Setup(i.e., GS-TBK.GKeyGen), Request (i.e., GS-TBK.Join), Visa (i.e., GS-TBK.Issue), Pass (i.e., GS-TBK.Sign), Verify, Revoke and Trace are 43.2/4.9/5.7 ms, 52.1/60.5/56.3 ms, 9.5/1.7/8.2 ms, 19.8/159.1/133.6 ms, 53.9/177.5/100.8 ms, 4.9/0.76/6.48 ms, and 30.2/0.76/0.78 ms, respectively. Since the execution time of the Trace and Revoke algorithms are related to the size of the registration list and the revocation list respectively, the size of the registration list and the revocation list are set to constant (equal to 1) during the test.

As shown in Fig. 7 and Table 1, our most important contribution is to improve the execution efficiency of the Pass algorithm, which is more than 400%, 4000%, 3400% better than the systems in [6,8,9], respectively. Compared to the state-of-the-art transit pass system [9], our Pass algorithm requires only 6 exponentiations in $\mathbb{G}_1$, while the Pass algorithm in [9] requires 6 exponentiations in $\mathbb{G}_1$, 2 exponentiations in $\mathbb{G}_2$ and 1 bilinear map operation e. Both exponentiation in $\mathbb{G}_2$ and bilinear map operation e are more time-consuming than the exponentiation in $\mathbb{G}_1$. The time consumption of the Verify algorithm in our system is also lower than that in [6,8,9], while the execution times of our Setup, Revoke and Trace algorithms are roughly equivalent to those in [9]. We note that the execution times of our Request and Visa algorithms are significantly higher than those in [6,8,9], but these algorithms only need to be executed once for each new user.

In Fig. 8, we compare the storage and communication cost in our system with that in [6,8,9] at $n = 10$. The size of the revocation of AnoPas is the same as the system in [9], but larger than the systems in [6,8]. The sizes of the group public key and signing key in our system are larger than those in [6,8,9], but only single copies of them need to be stored for each new user. The group signature size of this scheme is 20%, 80%, and 60% smaller than that of systems in [6,8,9], respectively, which effectively reduces the communication complexity of Pass algorithm.

5.2. Implementation on a smart card

Smart cards generally provide a high-level programming API with standard cryptography algorithms, basic arithmetic operations, and storage operations. However, the existing smart cards cannot support bilinear map operations. We implemented our AnoPas system on Aisinochip's state-of-the-art smart card chip - ACH512 [27] because it provides API access to SHA256, group operations over a prime field, and point multiplications on the elliptic curve as required in our passing protocol. Another reason is that the clock frequency of smart cards is between 10 MHZ - 100 MHz due to their small size and low power consumption. The clock frequency of ACH512 can be set between 6 MHz and 110 MHz; the higher the clock frequency, the faster the computation speed. In order to compare with other widely deployed

Table 5
Implementation Environment.

Participant	Hardware	CPU	Frequency	RAM	Operating system
Smart card	Aisinochip	ACH512	40 MHz	128 KB	COS
Mobile phone	HUAWEI Honor 9i	Hisilicon kirin 659 (ARMv8-A)	2.36 GHz and 1.7 GHz	4 GB	Android 9.0
PC	HUAWEI Matebook 14	AMD Ryzen-5 4600H	3.00 GHz	16 GB	Ubuntu Kylin 16.04

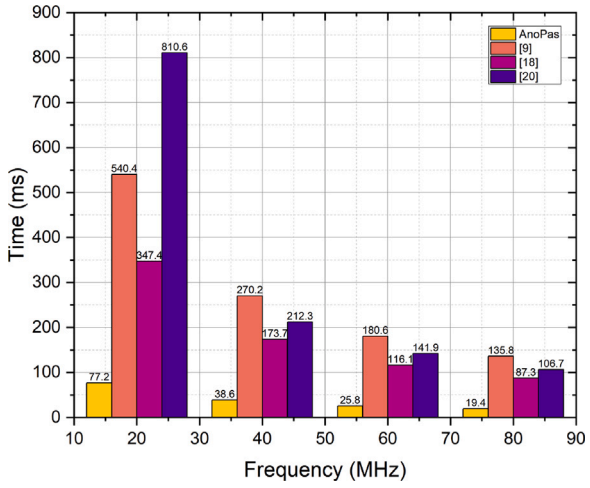

Fig. 10. Execution time of Pass algorithm on a smart card after precomputations.

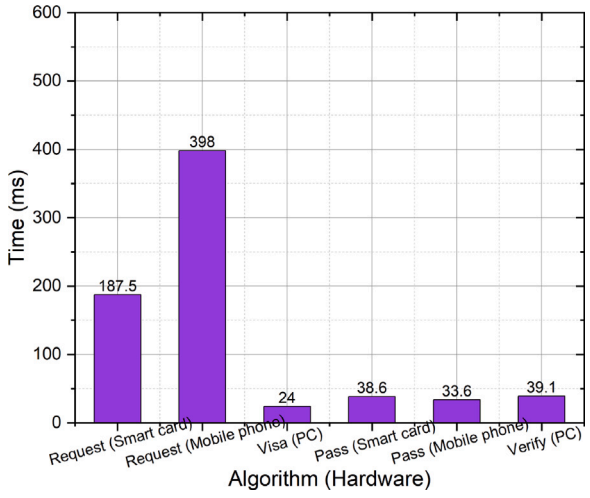

Fig. 11. Execution time on the mobile phone with smart card.

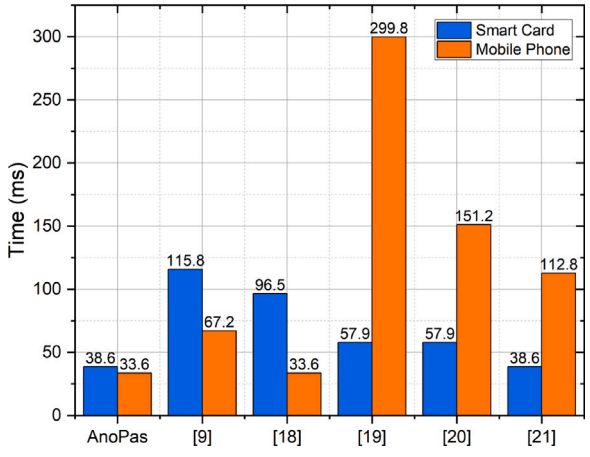

Fig. 12. Execution time of Pass algorithm on the mobile phone with smart card.

smart card products, we set the clock frequency of ACH512 to 20 MHz, 40 MHz, 60 MHz, and 80 MHz in our experiments.

In Fig. 9 we compare the computational cost of the Pass algorithm in our systems with that in [9,18,20]. To compare with the system in [9], we use a standard technique [28,29] by adding an element in $\mathbb{G}_1$ and a signature of knowledge to avoid using bilinear map operations on smart cards to implement the passing protocol in [9]. Since the schemes in [6,8,19,21] inevitably need to perform bilinear pairing operations or exponentiations in $\mathbb{G}_2$, making them impossible to implement on smart cards, we do not include them in this comparison. Our performance is significantly better than the pass protocol implementations in [9,18,20], as shown in Fig. 9. Specifically, our performance is more than 130%, 50%, and 250% better than [9,18,20], respectively.

Our Pass algorithm can be divided into two parts: the offline part and the online part. The offline part performs the Derive algorithm of the tag-based signature, which requires 4 exponentiations in $\mathbb{G}_1$ and it can be computed in advance. The online part performs the signature of knowledge of user's private key, which requires 2 exponentiations in $\mathbb{G}_1$ computed in real-time. In Fig. 10, when the precomputation is completed, we compare the computation cost of the Pass algorithm in our system with that in [9,18,20]. After precomputation, our performance is more than 600%, 350%, and 950% better than the implementations in [9,18,20], respectively.

5.3. Implementation on the mobile phone with smart card

We evaluated the performance of our AnoPas system in the mobile phone with smart card using an Aisinochip's smart card chip (ACH512) and a HUAWEI's mobile phone (Honor 9i) as the user, and a HUAWEI's PC (Matebook 14) as the authority and transport service. In Table 5, we show the setup of the implementation environment.

Fig. 11 shows the execution time of our system where ACH512's API is used to implement the operations on the smart card, MIRACL [11] is used to implement the operations on the mobile phone and PC, and $n = 10$ is chosen. Our Pass algorithm takes 38.6 ms on the smart card and 33.6 ms on the mobile phone. Our Request algorithm takes 187.5 ms on the smart card and 398 ms on the mobile phone. Our Visa and Verify algorithm take 24 ms and 39.1 ms on the PC, respectively.

Additionally, we conduct a comparative analysis of the computational costs associated with the Pass algorithm in our systems and those in previous works [9,18–21] across mobile phone and smart card platforms. As illustrated in Fig. 12, our system demonstrates markedly superior performance compared to the Pass protocol implementations detailed in [9,18–21]. Specifically, when considering computation time on smart cards and mobile phones, AnoPas exhibits a substantial advantage, being only 39%, 55%, 20%, 34%, and 47% of the computational cost of systems described in [9,18–21], respectively.

The above analysis and comparison show that our AnoPas system has significantly lower communication and computation overheads compared to state-of-the-art systems.

6. Conclusion

In this paper, we introduce AnoPas, a novel and practical anonymous transit pass system based on group signatures with time-bound keys (GS-TBK). We rigorously demonstrate its robust security, encompassing anonymity, traceability, and non-frameability. AnoPas substantially enhances the efficiency of transit pass protocols through the innovative GS-TBK scheme. We provide detailed implementations of AnoPas on various platforms, including PCs, smart cards, and mobile phones with smart cards, illustrating its suitability for real-world applications.

CRediT authorship contribution statement

Rui Shi: Conceptualization, Formal analysis, Software, Writing – original draft. **Yang Yang:** Conceptualization, Methodology, Writing – original draft, Funding acquisition. **Yingjiu Li:** Investigation, Supervision, Writing – review & editing, Methodology. **Huamin Feng:** Formal analysis, Software, Validation, Writing – review & editing. **Hwee Hwa Pang:** Supervision, Validation, Writing – review & editing. **Robert H. Deng:** Formal analysis, Validation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

Rui Shi and Huamin Feng are supported by Fundamental Research Funds for the Central Universities of China under Grant No. 3282023007, Beijing Natural Science Foundation, China under Grant No. 4234084. Yang Yang is supported by National Natural Science Foundation of China under Grant No. 62372110, Fujian Provincial Natural Science of Foundation, China under Grant No. 2023J02008.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.sysarc.2024.103184.

References

[1] Île-de-France Mobilités, Navigo, 2021, https://www.iledefrance-mobilites.fr/titres-et-tarifs/liste?d=forfaits.
[2] Metro, EZ transit pass, 2021, https://www.metro.net/riding/fares/ez-transit-pass/.
[3] N. Desmoulins, R. Lescuyer, O. Sanders, J. Traoré, Direct anonymous attestations with dependent basename opening, in: International Conference on Cryptology and Network Security, Springer, 2014, pp. 206–221.
[4] B. Libert, F. Mouhartem, K. Nguyen, A lattice-based group signature scheme with message-dependent opening, in: International Conference on Applied Cryptography and Network Security, Springer, 2016, pp. 137–155.
[5] D. Chaum, E. van Heyst, Group signatures, EUROCRYPT'91 547 (1991) 257–265.
[6] C.K. Chu, J.K. Liu, X. Huang, J. Zhou, Verifier-local revocation group signatures with time-bound keys, in: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, 2012, pp. 26–27.
[7] J.K. Liu, C.-K. Chu, S.S. Chow, X. Huang, M.H. Au, J. Zhou, Time-bound anonymous authentication for roaming networks, IEEE Trans. Inf. Forensics Secur. 10 (1) (2014) 178–189.
[8] K. Emura, T. Hayashi, A. Ishida, Group signatures with time-bound keys revisited: a new model, an efficient construction, and its implementation, IEEE Trans. Dependable Secure Comput. 17 (2) (2017) 292–305.
[9] O. Sanders, Improving revocation for group signature with redactable signature, in: IACR International Conference on Public-Key Cryptography, Springer, 2021, pp. 301–330.
[10] C. Hébant, D. Pointcheval, Traceable constant-size multi-authority credentials, 2020, Cryptology ePrint Archive.
[11] MIRACL Ltd, MIRACL, 2019, https://github.com/miracl/MIRACL.
[12] D. Boneh, X. Boyen, Short signatures without random oracles, in: International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2004, pp. 56–73.
[13] H.Y. Lin, W.G. Tzeng, An efficient solution to the millionaires' problem based on homomorphic encryption, in: International Conference on Applied Cryptography and Network Security, Springer, 2005, pp. 456–466.
[14] M.H. Au, W. Susilo, Y. Mu, Constant-size dynamic k-TAA, in: International Conference on Security and Cryptography for Networks, Springer, 2006, pp. 111–125.
[15] K. Ohara, K. Emura, G. Hanaoka, A. Ishida, K. Ohta, Y. Sakai, Shortening the libert-peters-yung revocable group signature scheme by using the random oracle methodology, IEICE Trans. Fundam. Electron. Commun. Comput. Sci. 102 (9) (2019) 1101–1117.
[16] T. Nakanishi, N. Funabiki, Revocable group signatures with compact revocation list using accumulators, IEICE Trans. Fundam. Electron. Commun. Comput. Sci. 98 (1) (2015) 117–131.
[17] J. Fan, F. Vercauteren, I. Verbauwhede, Faster Fp arithmetic for cryptographic pairings on Barreto-Naehrig curves, in: International Workshop on Cryptographic Hardware and Embedded Systems, Springer, 2009, pp. 240–253.
[18] J. Camenisch, M. Drijvers, A. Lehmann, Anonymous attestation using the strong diffie hellman assumption revisited, in: International Conference on Trust and Trustworthy Computing, Springer, 2016, pp. 1–20.
[19] J. Camenisch, M. Drijvers, A. Lehmann, Universally composable direct anonymous attestation, in: Public-Key Cryptography–PKC 2016, Springer, 2016, pp. 234–264.
[20] J. Camenisch, L. Chen, M. Drijvers, A. Lehmann, D. Novick, R. Urian, One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation, in: 2017 IEEE Symposium on Security and Privacy, SP, IEEE, 2017, pp. 901–920.
[21] L. Hanzlik, D. Slamanig, With a little help from my friends: constructing practical anonymous credentials, in: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, 2021, pp. 2004–2023.
[22] S.D. Galbraith, K.G. Paterson, N.P. Smart, Pairings for cryptographers, Discrete Appl. Math. 156 (16) (2008) 3113–3121.
[23] M. Chase, A. Lysyanskaya, On signatures of knowledge, in: Annual International Cryptology Conference, Springer, 2006, pp. 78–96.
[24] A. Fiat, A. Shamir, How to prove yourself: Practical solutions to identification and signature problems, in: Conference on the Theory and Application of Cryptographic Techniques, Springer, 1986, pp. 186–194.
[25] J. Camenisch, M. Stadler, Efficient group signature schemes for large groups, in: Annual International Cryptology Conference, Springer, 1997, pp. 410–424.
[26] V. Shoup, Lower bounds for discrete logarithms and related problems, in: International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 1997, pp. 256–266.
[27] Aisinochip, ACH512, 2022, http://www.aisinochip.com/index.php/product/show/id/20/catid/127.html.
[28] D. Bernhard, G. Fuchsbauer, E. Ghadafi, N.P. Smart, B. Warinschi, Anonymous attestation with user-controlled linkability, Int. J. Inf. Secur. 12 (3) (2013) 219–249.
[29] A. Barki, N. Desmoulins, S. Gharout, J. Traoré, Anonymous attestations made practical, in: Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks, 2017, pp. 87–98.