Token-Based Key Exchange - Non-Interactive Key Exchange meets Attribute-Based Encryption

Elsie Mestl Fondevik[1,2] and Kristian Gjøsteen[2]

[1] Kongsberg Defence & Aerospace, Norway
[2] Norwegian University of Science and Technology, Trondheim, Norway

Abstract. In this paper we define the novel concept *token-based key exchange (TBKE)*, which can be considered a cross between *non-interactive key exchange (NIKE)* and *attribute-based encryption (ABE)*. TBKE is a scheme that allows users within an organization to generate shared keys for a subgroup of users through the use of personal tokens and secret key. The shared key generation is performed locally and no interaction between users or with a server is needed.
The personal tokens are derived from a set of universal tokens and a master secret key which are generated and stored on a trusted central server. Users are only required to interact with the server during setup or if new tokens are provided. To reduce key escrow issues the server can be erased after all users have received their secret keys. Alternatively, if the server is kept available TBKE can additionally provide token revocation, addition and update.
We propose a very simple TBKE protocol using bilinear pairings. The protocol is secure against user coalitions based upon a novel *hidden matrix problem*. The problems requires an adversary to compute where the adversary must compute a matrix product in the exponent, where some components are given in the clear and others are hidden as unknown exponents. We argue that the hidden matrix problem is as hard as dLog in the bilinear group model.

Keywords: Non-interactive key exchange · Attribute-based encryption · Broadcast encryption

1 Introduction

Consider an organization such as a university. All university members; professors, students, teaching assistants and administrative staff belong to the same organization, but they do not, nor should they, have access to the same information and sub-domains. For example, exam solutions for a course should only be accessible to professors and teaching assistants (TAs) in that specific course. In particular, students should not be able to use their role as teaching assistant in one course to gain access to exam solutions for a different course. Even if a student convinces a professor in a different course to help them cheat, such a coalition should not succeed.

By introducing attributes such as *professor*, *student*, *TA*, *cryptography* and *CRYPTO101*, etc. students and employees alike can be given appropriate tokens. In the above example only people with access to both *professor* **and** *CRYPTO101* token should be able to access the CRYPTO101 exam solution. A *professor* in *BITCOIN101* collaborating with a *student* taking *CRYPTO101* should not be able to gain access to the CRYPTO101 exam solution, even though the union of their tokens matches the requirements.

E-mail: elsie.fondevik'@'kongsberg.com (Elsie Mestl Fondevik), kristian.gjosteen'@'ntnu.no (Kristian Gjøsteen)

The scheme proposed in this paper can be viewed as a merge between *non-interactive key exchange (NIKE)* [FHKP13, CKS08, DH76] and *attribute based encryption (ABE)* [GPSW06, SW05], where multiple users are able to produce a shared secret key based upon internal data (tokens) without interacting. We differentiate between an **attribute**, a value that is incorporated into something else, e.g. a key or ciphertext, and a **token**, a value in and of itself. The proposed scheme is denoted *token-based key exchange (TBKE)*.

TBKE only requires a single back and forth interaction with a trusted central server for each user during an initial setup. Once all members have been added and tokens and keys are distributed the server may be erased to reduce key escrow issues. Alternatively, if the server is kept running, new tokens can be provided and old ones revoked or updated.

The goal of TBKE is threefold. First, to reduce the number of distributed symmetric keys needed within a domain. In an extreme scenario an organization consisting of n users would require 2^n symmetric keys to be able to communicate in any possible subgroup. Through tokens the TBKE scheme can reduce this number down to as few as n tokens.

Example 1. In an organization consisting of n users and n tokens, number both tokens and users 1 though n. Distribute the tokens so that user i gets access to all tokens except token number i. Then, to create a shared key for a subset of users U, use all tokens $i \in \{1, 2, \cdots n\} \setminus U$.

In the case of pre-shared keys, the second goal of a TBKE scheme is to reduce the amount of planning needed when setting up a domain. If, instead, the keys are exchanged using a group key exchange protocol [ACDT20, CCG+18, BBR+23] TBKE drastically reduces transmission overhead.

The third benefit is the removal of membership knowledge. That is, users are no longer required to know all user-identities to compute a shared key when using a TBKE protocol. This is useful when the target audience is large and independent but still has clear mutual interests, e.g. conference participants. We would like to note that TBKE does not prevent membership knowledge, it only removes the strict requirement. Furthermore, authentication on sent messages may still be included at the discretion of the symmetric protocol used for message encryption.

1.1 General Construction

A TBKE scheme starts by setting up a trusted central server. The server produces a set of **universal tokens** and a single **master secret key**. These values function as a backdoor and blueprint for all secret information that is later distributed to individual users. It is important to note that neither the master key nor the universal tokens should ever be shared or leave the central server.

After server setup, users may be added to the organization. A user is given a distinct set of **personal tokens** as well as a unique **personal secret key**. The personal tokens are derived in correspondence to the users *access rights*.

A user selects a subset of personal tokens and together with its secret key derives a shared key. The set of personal tokens used for that specific key generation determines the shared key's *access restriction*. The shared key can then be used for message encryption. Only users with personal tokens that encompass the access restriction will be able to derive the correct shared key needed for decryption.

In order for a TBKE to be secure we require that a correctly derived shared key remains indistinguishable from a random value when at least one token is unknown. This requirement must hold true even if users decide to collaborate and pool all their secret keys and personal tokens, as long as none of the users can compute the shared key individually.

1.2 Proposed Protocol

We propose a matrix-based protocol. The protocol is simple and elegant in its design, and requires no harder operations than matrix multiplication and possibly a single bilinear pairing to compute personal information or a shared secret key.

The universal tokens are pairwise random, invertible matrices, with a fixed ordering. The server derives personal tokens as follows: For each new user added to the organization an initial invertible matrix is selected at random. Then for each universal token select further random invertible matrices. The first personal token is computed by multiplying the inverse of the initial random invertible matrix with the first universal token. This product is then multiplied with the random invertible matrix for the next token. A similar process happens for the remaining personal tokens where the previous token's random matrix is used as the initial matrix. In essence a personal token hides its corresponding universal token by bracketing the universal token with random matrices.

The personal tokens are designed in such a way that the random matrices cancel when the personal tokens are multiplied in order, which forces the users to use their tokens in the predefined order. Furthermore, by taking advantage of the non-commutative properties of matrices we prevent users from pooling their values.

As a basis for the security argument we introduce the *hidden matrix* problem. The hardness assumption of the problem is based on the fact that an adversary must, in a bilinear group structure, compute a matrix product in the exponent when only having access to some of the factors. The rest of the factors are hidden in the exponent or by other matrices.

1.3 Related Work

This work is similar to non-interactive key exchange and attribute-based encryption. Previous effort put in to combining the two research areas resulted in protocols requiring substantially more complex techniques to achieve the desired goal compared to our protocol [TZL17].

Attribute based encryption has become increasingly popular in recent years, expanding into multiple different sub-categories. At its core, ABE is an asymmetric encryption scheme that allows users to fine-tune access control to encrypted data based on attributes instead of the traditional public key identifier. The attributes are inherently public. An access policy, based on these attributes, is integrated either into each user's secret key or into the ciphertext, resulting in two variants: ciphertext-policy (CP) ABE and key-policy (KP) ABE.

If we disregard the difference between a key exchange and an asymmetric encryption scheme, the handling and secrecy concerning attributes (tokens) remains as one significant difference between a general ABE scheme and the construction in this paper. Comparing the two schemes, TBKE has more of a symmetric flavor than its counterpart, where only participating members should be able to derive the keys. The requirement concerning public attributes is thus circumvented.

Attribute based key exchange [KKL+16, GBG10] is similar to TBKE in the same way key encapsulation is similar to NIKE. The general difference between a TBKE scheme and ABE concerning the secrecy surrounding attributes, however, also applies to attribute based key exchange. This can mainly be seen in the ciphertext generated during encapsulation [GBG10], which scales linearly to the number of attributes in the access structure.

Broadcast encryption [FN94, BGW05, KMW23] aims to allow a streaming service to broadcast its content while preventing users without a valid subscription from benefiting from the stream. In terms of use case, broadcast encryption is quite similar to TBKE. There are, however, a couple key differences. The first is that broadcast encryption requires

an online server connection, whereas in TBKE the need of a trusted centralized server during shared key generation is removed. The trusted centralized server will only be needed during admission of new users into the organization, i.e., distribution of personal tokens and personal secret keys. When using the protocol to produce shared keys the server is never queried. In other words, when running a TBKE protocol, the centralized server may be erased after all users with their accesses are added, minimizing the amount of trust needed to be placed on such servers.

The second difference between broadcast encryption and TBKE is in the roles users have. In TBKE any user may be the one to initiate communication on a specific shared key, while in broadcast encryption the roles are static. The streaming service will always be the one broadcasting while users try to decrypt.

Previous work converging on TBKE from a NIKE viewpoint generally evolves around expanding NIKE to group communication [BC94, DLL22]. Their focus has mainly been on generating keys for user-subgroups using their public keys and ids. No additional separation values are considered.

2 Prerequisites

Let $\lambda \in \mathbb{N}$ be a natural number. We use 1^λ to denote the string consisting of λ 1's concatenated.

Let S denote a set and $\mathcal{P}(S)$ the power set of S. Furthermore, let $s \leftarrow\!\$\ S$ denote the process where s is sampled from S uniformly at random. If $\mathcal{S}$ is a distribution over S then $s \leftarrow\!\$\ \mathcal{S}$ denotes the process where s is sampled from S with probability distribution $\mathcal{S}$. Let A be a probabilistic algorithm, and we write $s \leftarrow\!\$\ A(x_1, x_2, \cdots, x_n)$ to denote that s is the output of the algorithm when run on input $x_1, x_2, \cdots, x_n$.

We write vectors, $\mathbf{v} \in \mathbb{Z}_q^k$, in lowercase bold font, while matrices, $\mathbf{A} \in \mathbb{Z}_q^{l\times k}$, are written in uppercase bold font. The identity matrix in $\mathbb{Z}_q^{k\times k}$ is denoted $\mathbf{I}_k$.

Let $\mathcal{G}$ be a cyclic group with generator g and let $a \in \mathbb{Z}$ be a scalar. We then use the notation from [EHK+13] and write $[a] = g^a$ to denote the element in $\mathcal{G}$ that occurs by applying generator g a times. Consequently, if $\mathbf{A} = (a_{i,j}) \in \mathbb{Z}^{l\times k}$ is a matrix then $[\mathbf{A}] = (g^{a_{i,j}}) \in \mathcal{G}^{l\times k}$.

We extend this notation to bilinear pairing group structures. Let $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$ be cyclic groups of order q and let $e : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_3$ be a bilinear map. We assume that there are fixed generators $g_1 \in \mathcal{G}_1$, $g_2 \in \mathcal{G}_2$ and $g_3 \in \mathcal{G}_3$ and for any matrix $\mathbf{A} = (a_{i,j}) \in \mathbb{Z}^{l\times k}$ we use the notation $[\mathbf{A}]_u = (g_u^{a_{i,j}}) \in \mathcal{G}_u^{l\times k}$, for $u \in \{1, 2, 3\}$.

3 Security Model

Conceptually we can view *token-based key exchange (TBKE)* as a non-interactive group key exchange algorithm that utilizes attributes, represented as tokens, to provide access-restricted shared keys. A trusted central server is needed during setup, key generation and if desirable to produce new tokens. Users in the organization produce shared secret keys by combining their personal secret key with a selection of personal tokens. The selection of personal tokens used in the generation of a shared key determines the access restriction of all messages encrypted under the key.

The security model used for TBKE shares many similarities with both the model presented by Gorantla, Boyd, and Nieto [GBG10] for attribute based key exchange, as well as the security model from Freire et.al. [FHKP13] regarding NIKE.

Definition 1 (Token-Based Key Exchange)**.** A token-based key exchange protocol TBKE consists of three mandatory PPT algorithms **setup**, **keyGen** and **sharedKey** and two

optional PPT algorithms *newToken* and *distToken*. The algorithms have the following properties.

$\mathbf{setup}(1^\lambda, n) \to (\mathbf{pp}, msk, \{t_i\}_{i=1}^n)$
: The *setup* algorithm takes a security parameter 1^λ and the maximum number n of tokens as input. It outputs a set of system parameters, $\mathbf{pp}$, together with a master secret key msk and a sequence of n universal tokens $(t_1, t_2, \ldots, t_n)$.

$\mathbf{keyGen}(msk, \{t_i\}_{i=1}^n, \mathbf{pp}, \mathsf{id}, J) \to (\mathsf{sk}_{\mathsf{id}}, \{\tilde{t}_j^{(\mathsf{id})}\}_{j\in J})$
: The *key generation* algorithm takes the master secret key msk, universal tokens $\{t_i\}_{i=1}^n$, system parameters $\mathbf{pp}$, user id id and a list of access rights J as input. It outputs a personal private key $\mathsf{sk}_{\mathsf{id}}$ together with a list of personal tokens $\{\tilde{t}_j^{(\mathsf{id})}\}_{j\in J}$. The algorithm requires that $J \subseteq \{1, 2, \cdots, n\}$.

$\mathbf{sharedKey}(\mathsf{sk}_{\mathsf{id}}, \{\tilde{t}_j^{(\mathsf{id})}\}_{j\in J}, K) \to (\mathsf{k}_K)$
: The *shared key* algorithm takes the personal secret key $\mathsf{sk}_{\mathsf{id}}$, personal tokens $\{\tilde{t}_j^{(\mathsf{id})}\}_{j\in J}$ and access restriction K as input. The algorithm outputs either a shared key k, or $\perp$ if the algorithm failed. If $K \not\subseteq J$ then $\perp$ is returned.

$newToken(msk, \{t_i\}_{i=1}^n, \mathbf{pp}) \to (\mathbf{pp}', t_{n+1})$ **(Optional)**
: The *new token* algorithm takes the master secret key msk, a list of all universal tokens $\{t_i\}_{i=1}^n$ and the public parameters $\mathbf{pp}$ as input. It outputs updated public parameters $\mathbf{pp}'$, and a new token t_{n+1}.

$distToken(msk, \{t_i\}_{i=1}^n, \mathbf{pp}, \mathsf{id}, i) \to \left(\tilde{t}_i^{(\mathsf{id})}\right)$ **(Optional)**
: The *new personal token* algorithm takes the master secret key msk, universal tokens $\{t_i\}_{i=1}^n$, public parameter $\mathbf{pp}$, user id id and token id i as input. It outputs a personal token $\tilde{t}_i^{(\mathsf{id})}$ for user id based on token i. The algorithm requires that $i \in \{1, 2, \ldots, n\}$.

TBKE protocols that implement both optional algorithms are said to be *dynamic*, while protocols that implement neither are said to be *static*.

Let universal tokens and master secret be generated as $(\mathbf{pp}_0, msk, \{t_i\}_{i=1}^n) \leftarrow\$ \mathbf{setup}(1^\lambda, n)$ and $(\mathbf{pp}_l, t_{n+l}) \leftarrow\$ newToken(msk, \{t_i\}_{i=1}^{n+l-1}, \mathbf{pp}_{l-1})$, for $l = 1, 2, \ldots, s+1$ where s is the number of new tokens added after setup. Let $\mathsf{id}_1, \mathsf{id}_2$ be two identities, let $J_1, J_2 \subseteq \{1, 2, \ldots, n+s\}$, let $s_1, s_2 \in \{0, 1, \ldots, s\}$ and let $\tilde{J}_i = J_i \cap \{1, 2, \ldots, n+s_i\}$, $i = 1, 2$. Let, for $i = 1, 2$, the personal key material be generated as $(\mathsf{sk}_{\mathsf{id}_i}, \{\tilde{t}_j^{(\mathsf{id}_i)}\}_{j\in\tilde{J}_i}) \leftarrow\$ \mathbf{keyGen}(msk, \{t_j\}_{j=1}^{n+s_i}, \mathbf{pp}_{s_i}, \mathsf{id}_i, \tilde{J}_i)$ and $\tilde{t}_v^{(\mathsf{id}_i)} \leftarrow\$ distToken(msk, \{t_j\}_{j=1}^{n+v}, \mathbf{pp}_v, \mathsf{id}_i, v)$. Let $K \subseteq \{1, 2, \cdots, n+s\}$ be an access restriction such that $K \subseteq J_1$ and $K \subseteq J_2$. We say that **correctness** holds if

$$\mathbf{sharedKey}(\mathsf{sk}_{\mathsf{id}_1}, \{\tilde{t}_j^{(\mathsf{id}_1)}\}_{j\in J_1}, K) = \mathbf{sharedKey}(\mathsf{sk}_{\mathsf{id}_2}, \{\tilde{t}_j^{(\mathsf{id}_2)}\}_{j\in J_2}, K).$$

The public parameter $\mathbf{pp}$ include the number of tokens available in the protocol at all times. The public parameter is initially created during setup and further updated when new tokens are introduced with *newToken*.

The setup algorithm is run by a trusted server and produces a *master key msk* together with a list of *universal tokens* $(t_1, t_2, \cdots, t_n)$. Both the master secret key as well as the universal tokens are private information and are only ever known to the server. New tokens can be added by the server with the optional *newToken*-algorithm, which can be run anytime after setup to initiate the creation of additional tokens. If *newToken* is not implemented the number of tags n will be fixed during setup and not changed subsequently.

To add a new user to the organization the server is queried by the user with its desired access rights. The server generates a new personal secret key and personal tokens according

to the access right using **keyGen**. The personal secret key as well as the personal tokens are sent to the user. (Determining if a user query is valid in terms of access rights is up to the individual application to ensure that specific access policies are enforced.)

To expand a user's access rights, tokens can be distributed after the initial **keyGen**-algorithm was run by allowing users to query the server for additional tokens. The *distToken*-algorithm is run by the server and generates and outputs the specified personal token for the requesting user. The algorithm is optional. If the algorithm is not implemented a user's access rights will be fixed after added to the organization.

In order to derive a shared key k_K, users execute the **sharedKey** algorithm with their secret key and the personal tokens corresponding to K as input. The computation is performed locally by each user individually. Only those organization members with all the required tokens, as defined by K, should be able to correctly compute k_K. Note **sharedKey** is the only TBKE algorithm run by the users, all the other algorithms are run by a trusted central server.

Remark 1. We would like to emphasize the fact that if the optional algorithms are not implemented then all secret data stored on the server can be erased once all users are added to the organization. This reduces the amount of trust needed to be placed in the server in terms of honesty and secure storage. The drawback of erasing the server, however, is that any implemented TBKE protocol becomes static both in terms of available and accessible tokens as well as organization membership.

3.1 Security Notion

Informally, users should only be able to generate shared keys where they have all the required tokens. Computing a shared secret key when at least one token is unknown should not result in a meaningful value. Furthermore, coalitions of users should not gain any information nor access to more shared secret keys than they would individually.

This means that a computed shared key should not leak any other information than the access requirements. Moreover, personal tokens and secret keys should not leak information about the underlying universal tokens and master secret key, nor should combining multiple users personal tokens and secret keys leak any more information than they do individually.

The security experiment, Exp^{TBKE}, defined in Definition 2 runs a simulation of a TBKE protocol Π against a real-or-random adversary. The adversary can influence and gain information about the experiment by issuing specific queries. At one point it will request a challenge key for an access restriction K, and receive either the real shared key or a random key. The adversary wins if it correctly determines if the key was real or random. The adversary's advantage is defined in the usual way, as the distance from $1/2$ to the probability that an adversary will guess correctly indicates the security level of the protocol. The experiment is modular and can be used for both static and dynamic TBKE protocols.

Definition 2. Let Π be a TBKE scheme, and let Exp^{TBKE} be the experiment described in Figure 1. An adversary against Π interacts with the experiment. The advantage of an adversary $\mathcal{A}$ against Π is defined to be

$$\mathsf{Adv}^{\text{X-TBKE}}_{\Pi-\mathcal{A}}(\lambda) = 2\left|\Pr\left[\text{Exp}^{X-TKBE}_{\Pi-\mathcal{A}}(1^\lambda, n)\right] - \frac{1}{2}\right|,$$

where $X \in \{\text{dynamic, static}\}$.

Remark 2. Since personal tokens and secret keys are all derived from the universal tokens and the master key, the experiment has full access to all private information. Due to the correctness requirement of any TBKE protocol all personal secret keys and tokens should produce the same shared key when computed on corresponding input. As such we allow

$\text{EXP}^{X-TBKE}_{\Pi-\mathcal{A}}(1^\lambda, n)$

1 : $\mathcal{C} = \emptyset;\ \mathsf{CR}[\cdot] = \emptyset$
2 : $b \leftarrow\!\$\ \{0,1\}$
3 : $(\mathbf{pp}, msk, \{t_i\}_{i=1}^n) \leftarrow\!\$\ \mathbf{setup}(1^\lambda, n)$
4 : $K \leftarrow\!\$\ \mathcal{A}^X(n)$
5 : $\mathsf{k}_{K,0} \leftarrow\!\$\ \mathbf{sharedKey}(msk, \{t_i\}_{i=1}^n, K)$
6 : $\mathsf{k}_{K,1} \leftarrow\!\$\ \mathcal{K}$
7 : $b' \leftarrow\!\$\ \mathcal{A}^X(\mathsf{k}_{K,b})$
8 : $\mathcal{A}$ wins if $b = b'$ and $K \notin \mathcal{C}$.

ORACLE $\mathcal{O}_{\mathbf{addToken}}()$

1 : **Require** X = dynamic
2 : $\mathbf{pp}, t_{n+1} \leftarrow\!\$\ newToken(msk, \{t_i\}_{i=1}^n, \mathbf{pp})$
3 : $\{t_i\}_{i=1}^n \xleftarrow{\cup} \{t_{n+1}\}$
4 : **return pp**

ORACLE $\mathcal{O}_{\mathbf{sharedKey}}(J)$

1 : $\mathcal{C} \xleftarrow{\cup} \{J\}$
2 : **return sharedKey**$(msk, \{t_i\}_{i=1}^n, J)$

ORACLE $\mathcal{O}_{\mathbf{revealUser}}(\mathsf{id}, J)$

1 : **Require** $J \subseteq \{1, 2, \cdots, n\}$
2 : **if** $\mathsf{CR}[\mathsf{id}] = \emptyset$
3 : $\quad(\mathsf{sk}_{\mathsf{id}}, \{\tilde{t}_j^{(\mathsf{id})}\}_{j\in J}) \leftarrow\!\$\ \mathbf{keyGen}(msk, \{t_i\}_{i=1}^n, \mathbf{pp}, \mathsf{id}, J)$
4 : $\quad\mathsf{CR}[\mathsf{id}] \leftarrow (\mathsf{sk}_{\mathsf{id}}, \{\tilde{t}_j^{(\mathsf{id})}\}_{j\in J})$
5 : **else**
6 : $\quad(\mathsf{sk}_{\mathsf{id}}, \{\tilde{t}_j^{(\mathsf{id})}\}_{j\in \tilde{J}}) \leftarrow \mathsf{CR}[\mathsf{id}]$
7 : $\quad$**Require** $\left(\tilde{J} \subseteq J\right) \wedge$ (X = dynamic)
8 : $\quad$**for** $l \in J \setminus \tilde{J}$
9 : $\quad\quad\tilde{t}_l^{(\mathsf{id})} \leftarrow\!\$\ distToken(msk, \{t_i\}_{i=1}^n, \mathbf{pp}, \mathsf{id}, l)$
10 : $\quad\quad\mathsf{CR}[\mathsf{id}] \leftarrow (\mathsf{sk}_{\mathsf{id}}, \{\tilde{t}_j^{(\mathsf{id})}\}_{j\in\tilde{J}} \cup \{\tilde{t}_l^{(\mathsf{id})}\}$
11 : $\mathcal{C} \xleftarrow{\cup} \mathcal{P}(J)$
12 : **return** $\mathsf{CR}[\mathsf{id}]$

Figure 1: TBKE security experiment for both static and dynamic TBKE. The set $\mathcal{C}$ indicates a collection of revealed personal token-sets, while CR indicates generated and revealed users.

for abuse of notation and permit algorithm **sharedKey** to be run on universal tokens and the master secret key in place of personal tokens and secret key.

A TBKE adversary against $\text{Exp}_{\Pi}^{X-TBKE}$ can query $\mathcal{O}_{\textbf{revealUser}}$ to gain access to personal tokens and secret keys. The adversary sends a user id as well as a desired set of access rights as input to the query. If the user has not previously been queried, the oracle will compute personal tokens and the secret key according to **keyGen**. If the user has already been revealed, and Π is dynamic, the necessary additional personal tokens will be derived. If Π is static no further computation occurs after the first initial query. The resulting personal secret key and personal tokens will be returned to the adversary.

To gain access to shared secret keys it would normally not be able to compute, the adversary has access to oracle $\mathcal{O}_{\textbf{sharedKey}}$. The adversary sends an access restriction as input to the oracle and receives a shared key with the specified access restriction in return. The shared key is generated in accordance to the **sharedKey** algorithm. The oracle does not reveal the secret information needed for computation, only the resulting computed key.

In the case where Π is a dynamic TBKE protocol the adversary has access to one additional oracle; $\mathcal{O}_{\textbf{addToken}}$. This oracle takes no input and generates one universal token per query. It returns an updated public parameter to the adversary. The generated token is not revealed.

At some point the adversary can ask the experiment for a challenge on an access restriction K. The adversary will then receive a shared secret key $\mathsf{k}_{K,b}$ in return. The goal of the adversary is to determine if $\mathsf{k}_{K,b}$ is *real or random (ROR)*. The adversary wins if it correctly determines the validity of the shared secret key. To prevent the adversary from trivially winning we require that the access restriction K is not contained in an access right the adversary has revealed through $\mathcal{O}_{\textbf{revealUser}}$ or queried directly to $\mathcal{O}_{\textbf{sharedKey}}$.

4 Hidden Matrices and a TBKE Protocol

The final part of the paper describes a TBKE protocol that we later argue to be secure. The protocol comes in both a static and dynamic variant.

To simplify the construction, we expand upon the concept of tokens. It is important to note that all alterations to the tokens could be defined as part of the personal secret keys, but for presentational purposes and readability it is incorporated into an extended token. In other words, the alteration is cosmetic, not functional.

The change we introduce is splitting all tokens into a yes- and no-part. The yes-part of the token is distributed as before to users depending on the specific users' access rights. The no-part of the token, however, is distributed to all users independent of access rights. In other words, a user with access rights J will gain yes-tokens t_j^{yes} for all $j \in J$ and no-tokens t_j^{no} for all $j \in \{1, 2, \cdots, n\}$. Then to generate a shared key k_K, all yes-tokens corresponding to the access restriction K are selected. For the remaining values $\{1, 2, \cdots, n\} \setminus K$ the corresponding no-tokens are used.

It should be clear that yes-tokens work as regular tokens where shared key generation depends on specific users access rights. The no-tokens, on the other hand, are distributed to all and play more of a "filler" role. They are, however, personal.

4.1 Protocol Construction

For protocol construction we will focus on the static version. A description is later given of how to expand the static TBKE protocol to a dynamic variant. The full static protocol can be found in Figure 2.

In the construction, each universal token $t_i = (\mathbf{T}_i^{yes}, \mathbf{T}_i^{no})$ consists of two independent random invertible $(k \times k)$-matrices. From the universal tokens the personal tokens are created by multiplying the universal token with matrices on both sides. The matrices are

setup$(1^\lambda, n)$

1 : **for** $i = 1, 2, \cdots, n$:
2 : $\quad \mathbf{T}_i^{yes}, \mathbf{T}_i^{no} \leftarrow\$ \mathcal{D}_{k,k}$
3 : $\quad t_i \leftarrow (\mathbf{T}_i^{no}, \mathbf{T}_i^{yes})$
4 : $\mathbf{L} \leftarrow\$ \mathcal{D}_{l,k}$
5 : $\mathbf{R} \leftarrow\$ \mathcal{D}_{k,m}$
6 : $msk \leftarrow (\mathbf{L}, \mathbf{R})$
7 : **return** $\mathbf{pp}, msk, \{t_i\}_{i=1}^n$

sharedKey$(\mathsf{sk}_{\mathsf{id}}, \{\tilde{t}_i^{(\mathsf{id})}\}_{i=1}^n, K)$

1 : $\left(\mathbf{pp}, [\mathbf{L}\mathbf{A}_0]_1, \left[\mathbf{A}_n^{-1}\mathbf{R}\right]_2\right) \leftarrow \mathsf{sk}_{\mathsf{id}}$
2 : $\mathbf{T}_K \leftarrow \prod_{i=1}^{n} \mathbf{T}_{\mathsf{id},i}^{f(i,K)}$
3 : $\mathsf{k}'_K \leftarrow e([\mathbf{L}\mathbf{A}_0\mathbf{T}_K]_1, \left[\mathbf{A}_n^{-1}\mathbf{R}\right]_2)$
4 : $\mathsf{k}_K \leftarrow H\left(K, k'_K\right)$
5 : **return** k_K

keyGen$(msk, \{t_i\}_{i=1}^n, \mathbf{pp}, \mathsf{id}, J)$

1 : $(\mathbf{L}, \mathbf{R}) \leftarrow msk$
2 : $\mathbf{A}_0, \mathbf{A}_1, \cdots, \mathbf{A}_n \leftarrow\$ \mathcal{D}_{k,k}$
3 : $\mathsf{sk}_{\mathsf{id}} \leftarrow \left(\mathbf{pp}, [\mathbf{L}\mathbf{A}_0]_1, \left[\mathbf{A}_n^{-1}\mathbf{R}\right]_2\right)$
4 : **for** $i = 1, 2, \cdots, n$:
5 : $\quad \mathbf{T}_{\mathsf{id},i}^{yes} \leftarrow \begin{cases} \mathbf{A}_{i-1}^{-1}\mathbf{T}_i^{yes}\mathbf{A}_i, & \textbf{if } i \in J \\ \mathbf{I}_k, & \textbf{else} \end{cases}$
6 : $\quad \mathbf{T}_{\mathsf{id},i}^{no} \leftarrow \mathbf{A}_{i-1}^{-1}\mathbf{T}_i^{no}\mathbf{A}_i$
7 : $\quad \tilde{t}_i^{(\mathsf{id})} \leftarrow (\mathbf{T}_{\mathsf{id},i}^{no}, \mathbf{T}_{\mathsf{id},i}^{yes})$
8 : **return** $\mathsf{sk}_{\mathsf{id}}, \{\tilde{t}_i^{(\mathsf{id})}\}_{i=1}^n$

Figure 2: A static-TBKE protocol description. The public parameters $\mathbf{pp}$ contain the appropriate bilinear pairing structure as well as the total number of tags currently in use. The function $f : \mathbb{Z} \times \mathcal{P}(\mathbb{Z}) \to \{yes, no\}$ takes a number j and a set J as input and returns yes if $j \in J$ and otherwise returns no. The distribution $\mathcal{D}_{l,k}$ is the uniform distribution over full rank matrices in $\mathbb{Z}_q^{l\times k}$. H is a hash function.

unique per user and cancel when yes- or no-tokens, from a single user, are multiplied in order. That is, the personal token $\tilde{t}_i^{(\mathsf{id})}$ for user with user id id takes the form $\tilde{t}_i^{(\mathsf{id})} = (\mathbf{T}_{\mathsf{id},i}^{yes}, \mathbf{T}_{\mathsf{id},i}^{no})$ where $\mathbf{T}_{\mathsf{id},i}^{x} = \mathbf{A}_{i-1}^{-1}\mathbf{T}_i^{x}\mathbf{A}_i,\ x \in \{yes, no\}$ and where the matrices $\mathbf{A}_i \in \mathbb{Z}^{k\times k}$ are random, invertible and user-specific.

The master secret key $msk = (\mathbf{L}, \mathbf{R})$ consist of two non-square matrices taken over cyclic groups $\mathcal{G}_1$ and $\mathcal{G}_2$ respectively, i.e., $\mathbf{L} \in \mathcal{G}_1^{l\times k}$ and $\mathbf{R} \in \mathcal{G}_2^{k\times m}$. The secret key, $\mathsf{sk}_{\mathsf{id}} = (\mathbf{L}_{\mathsf{id}}, \mathbf{R}_{\mathsf{id}})$, for user id, is also a pair of non square matrices. The two matrices $\mathbf{L}_{\mathsf{id}}$ and $\mathbf{R}_{\mathsf{id}}$ are generated from the master secret key using the left and right cancellation matrix for tokens $\tilde{t}_0^{(\mathsf{id})}$ and $\tilde{t}_n^{(\mathsf{id})}$, respectively.

To generate a shared key k_K, yes-tokens for $j \in K$ and no-tokens for $j \notin K$ are multiplied and used as an exponent with the left matrix of $\mathsf{sk}_{\mathsf{id}}$ as base. The right matrix is incorporated using a bilinear pairing $e : \mathcal{G}_1^{l\times k} \times \mathcal{G}_2^{k\times m} \to \mathcal{G}_3^{l\times m}$ that maps $([\mathbf{U}]_1, [\mathbf{V}]_2) \mapsto [\mathbf{U}\mathbf{V}]_3$, where $\mathbf{U} \in \mathbb{Z}^{l\times k}$ and $\mathbf{V} \in \mathbb{Z}^{k\times m}$.

Theorem 1. *Let TBKE be the protocol described in Figure 2. Then TBKE is a token based key exchange protocol.*

Proof. We need to show that TBKE satisfies the syntactical requirements and that correctness holds. As discussed in the beginning of the section, all changes to the original protocol definition are purely cosmetic and as such the syntactical requirement follows directly from Definition 1. What remains to show is correctness.

Let $\mathsf{id}_1, \mathsf{id}_2$ be the user ids of two arbitrary users in the same organization. Let $J_1, J_2 \subseteq \{1, 2, \cdots, n\}$ be two arbitrary access rights. Let $(msk, \mathbf{pp}, (t_i)_{i=1}^n) \leftarrow\$ \mathbf{setup}(1^\lambda, n)$, $(\mathsf{sk}_{\mathsf{id}_x}, \{\tilde{t}_j^{(\mathsf{id}_x)}\}_{j\in J_x}) \leftarrow\$ \mathbf{keyGen}(msk, \mathbf{pp}, \mathsf{id}_x, J_x)$ for $x \in \{1, 2\}$ be the personal tokens and secret keys generated for the two users id_1 and id_2. Then for any arbitrary access right

K where $K \subseteq J_1 \cap J_2$, the shared key computed by user id_x is a follows

$$\begin{aligned}
\mathbf{sharedKey}(\mathsf{sk}_{\mathsf{id}_x}, \{\tilde{t}_j^{(\mathsf{id}_x)}\}_{j\in J_x}, K) &= H\left(K, e\left(\left[\mathbf{L}\mathbf{A}_{\mathsf{id}_x,0}\mathbf{T}_K\right]_1, \left[\mathbf{A}_{\mathsf{id}_x,n}^{-1}\mathbf{R}\right]_2\right)\right) \\
&= H\left(K, e\left(\left[\mathbf{L}\mathbf{A}_{\mathsf{id}_x,0}\prod_{i=1}^{n}\mathbf{T}_{\mathsf{id}_x,i}^{f(i,K)}\right]_1, \left[\mathbf{A}_{\mathsf{id}_x,n}^{-1}\mathbf{R}\right]_2\right)\right) \\
&= H\left(K, e\left(\left[\mathbf{L}\mathbf{A}_{\mathsf{id}_x,0}\prod_{i=1}^{n}\mathbf{A}_{\mathsf{id}_x,i-1}^{-1}\mathbf{T}_i^{f(i,K)}\mathbf{A}_{\mathsf{id}_x,i}\right]_1, \left[\mathbf{A}_{\mathsf{id}_x,n}^{-1}R\right]_2\right)\right) \\
&= H\left(K, e\left(\left[\mathbf{L}\prod_{i=1}^{n}\mathbf{T}_i^{f(i,K)}\cdot\mathbf{A}_{\mathsf{id}_x,n}\right]_1, \left[\mathbf{A}_{\mathsf{id}_x,n}^{-1}\mathbf{R}\right]_2\right)\right) \\
&= H\left(K, \left[\mathbf{L}\prod_{i=1}^{n}\mathbf{T}_i^{f(i,K)}\cdot\mathbf{A}_{\mathsf{id}_x,n}\mathbf{A}_{\mathsf{id}_x,n}^{-1}\mathbf{R}\right]_3\right) \\
&= H\left(K, \left[\mathbf{L}\prod_{i=1}^{n}\mathbf{T}_i^{f(i,K)}\mathbf{R}\right]_3\right)
\end{aligned}$$

It is clear that as long as the users id_1 and id_2 both have access rights that contain K then

$$\mathbf{sharedKey}(\mathsf{sk}_{\mathsf{id}_1}, \{\tilde{t}_j^{(\mathsf{id}_1)}\}_{j\in J_1}, K) = \mathbf{sharedKey}(\mathsf{sk}_{\mathsf{id}_2}, \{\tilde{t}_j^{(\mathsf{id}_2)}\}_{j\in J_2}, K)$$

and the correctness requirement holds. □ □

4.1.1 Optional dynamic expansion.

The protocol can be expanded to allow for dynamic token generation and distribution. The dynamic property introduced gives rise to additional properties such as token revocation and update.

The dynamic expansion is based upon the fact that if a user does not have access to multiple tokens in a row the no-tokens for those instances can be combined to reduce space without losing functionality. Taking advantage of this fact, the inverse can be utilized to create additional tokens.

The expansion functions by never distributing a personal yes-token for the last universal token. By never distributing the yes-token for the final token we have in essence never created the last universal yes-token. This last universal token can then be arbitrarily factorized to create a new token. By always leaving the last token unfulfilled an unbounded number of tokens can be created.

If users then come to an agreement to discontinue a specific token, the corresponding (personal) yes-token can be erased. The remaining no-token can then be compressed, by multiplying it, into one of the tokens on either side. This compression keeps the number of tokens stored at a minimum. It is here important that a leftover no-token is multiplied into both the no- and yes-token (if it exists). The tokens can be renumbered if desired, or the original numbering can be kept to prevent ambiguity.

If corrupted users are detected, an old token can be revoked and a new token can be generated and distributed to all users, with the exception of the corrupted user. This efficiently and effectively excludes the user from the protocol.

4.2 Security

We are now ready to look at the security of the proposed protocol. It is clear that if the personal tokens and secret keys leak information about the underlying universal tokens then the whole construction will break. The first step in determining the security of the

protocol is to ensure that no such leakage exists, even when multiple sets of personal tokens and keys are known. We will start by taking a step back and consider the relationship between universal tokens, personal tokens and the token-product generated as a sub-step when computing a shared key.

Universal tokens can be viewed as a $2n+2$ sequence of matrices, $\mathbf{L} \in \mathbb{Z}_q^{l\times k}$, $\mathbf{T}_1^{x_1}, \mathbf{T}_2^{x_2}, \ldots, \mathbf{T}_n^{x_n} \in \mathbb{Z}_q^{k\times k}$, $x_i \in \{yes, no\}$, $\mathbf{R} \in \mathbb{Z}_q^{k\times m}$, all of maximal rank. Let $e : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_3$ be a bilinear group structure where all the groups have order q. Then for $J \subseteq \{1, 2, \ldots, n\}$, and a sequence of invertible matrices $\mathbf{A}_0, \mathbf{A}_1, \ldots, \mathbf{A}_n$, we define $(J, \mathbf{A}_i)$-*reveal* over the universal tokens to be the sequence $[\mathbf{L}\mathbf{A}_0]_1 \in \mathcal{G}_1^{l\times k}$, $\mathbf{A}_{i-1}^{-1}\mathbf{T}_i^{no}\mathbf{A}_i \in \mathbb{Z}_q^{k\times k}$, $i = 1, 2, \ldots, n$, $\mathbf{A}_{i-1}^{-1}\mathbf{T}_i^{yes}\mathbf{A}_i \in \mathbb{Z}_q^{k\times k}$, $i \in J$, and $[\mathbf{A}_n^{-1}\mathbf{R}]_2 \in \mathcal{G}_2^{k\times m}$. In other words a $(J, \mathbf{A}_i)$-reveal creates personal tokens given an access right J and a predefined sequence of hiding matrices $\mathbf{A}_i$.

Combining the task of computing the token-product $[\mathbf{L}\prod_{i=1}^n \mathbf{T}_i^{f(i,J)}\mathbf{R}]_3$ with an adversary that may ask to reveal personal tokens gives rise to a new problem; the *general hidden matrix problem.*

Given a sequence of matrices, $\mathbf{L} \in \mathbb{Z}_q^{l\times k}$, $\mathbf{T}_1^{x_1}, \mathbf{T}_2^{x_2}, \ldots, \mathbf{T}_n^{x_n} \in \mathbb{Z}_q^{k\times k}$, $x_i \in \{yes, no\}$, $\mathbf{R} \in \mathbb{Z}_q^{k\times m}$, and a bilinear pairing $e : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_3$. An adversary against the *general hidden matrix problem* has access to a *reveal*-oracle that on input J_j samples a sequence of invertible matrices $\mathbf{A}_0, \mathbf{A}_1, \ldots, \mathbf{A}_n$ and returns a $(J_j, \mathbf{A}_{j,i})$-reveal of the hidden matrix sequence. At some point, the adversary outputs $J \subseteq \{1, 2, \ldots, n\}$ and eventually also an element $\mathbf{G} \in \mathcal{G}_3^{l\times m}$. The adversary *wins* if

$$\mathbf{G} = [\mathbf{L}\prod_{i=1}^{n} \mathbf{T}_i^{f(i,J)}\mathbf{R}]_3,$$

and $J \not\subseteq J_j$ for all reveal query inputs J_j. The *advantage* of the adversary is the probability that it wins.

We consider two variants of the general hidden matrix problem. In the *selective* variant, the adversary specifies the target J before making any oracle queries. In the *gap* variation, the adversary has access to a *gap*-oracle that on input of J' and an element of $\mathcal{G}_3^{l\times m}$ returns 1 if the group elements equal $[\mathbf{L}\prod_{i=1}^n \mathbf{T}_i^{f(i,J')}\mathbf{R}]_3$, and 0 otherwise.

We start by considering the relationship between the gap general hidden matrix problem and the construction from Figure 2.

Theorem 2. *Let $\mathcal{A}$ be a adversary against the construction from Figure 2. Then there exists an adversary $\mathcal{B}$ against the gap general hidden matrix problem with essentially the same run-time and the same advantage.*

Proof. The proof idea is to create an adversary $\mathcal{B}$ that uses $\mathcal{A}$ as an internal algorithm by simulating a run of Figure 2 to solve the problem. Since we operate in the random oracle model, the only way $\mathcal{A}$ can distinguish a shared key from a random value is if it queries the random oracle on specific inputs. Input which $\mathcal{A}$ necessarily needs to be able to compute. By using the gap oracle and keeping a log of all oracle queries, the adversary $\mathcal{B}$ is able to determine when and if the challenge is computed, and return this value.

Now for the construction of $\mathcal{B}$. In order for this construction to work, $\mathcal{B}$ must simulate oracles **revealUser**, **sharedKey** and the random-oracle used for hashing, in such a way that $\mathcal{B}$ is able to detect whenever the adversary makes the correct query to the random-oracle.

The **revealUser**-oracle is trivially simulated using reveal-oracle of the general hidden matrix problem.

To simulate the random-oracle and the **sharedKey**-oracle, we keep one list of random oracle queries, a list of gap oracle queries with their results and one list of known shared

keys. Then when **sharedKey**-oracle gets J', it first checks if J' is in the list of known shared keys and if so outputs the corresponding shared key. Otherwise, it samples a random key, records that key as a known shared key for J' and returns the key.

Recall that all queries to the random-oracle take the form J' together with an element in $\mathcal{G}_3^{l,k}$. The random-oracle is simulated by first forwarding the oracle query to the gap oracle. If the gap oracle returns 0, a new hash value is sampled and returned. If the gap oracle returns 1 and J' is on the list of known shared keys we instead use the known shared key as the hash value. Otherwise, we sample a hash value and record a known shared key for J' and the sampled hash value and return the hash value.

When adversary $\mathcal{A}$ specifies the target J, we let $\mathcal{B}$ select the same target and forward a "hashed"-version of the challenge to $\mathcal{A}$. If any subsequent gap oracle query involving J results in 1, let $\mathcal{B}$ output the query. We also go through the list of previous queries to see if any gap oracle queries involving J resulted in 1, in which case we output that query.

When $\mathcal{A}$ stops, $\mathcal{B}$ stops with random output.

By inspection, we see that our simulation of the oracles is perfect and requires negligible extra time. Furthermore, if $\mathcal{B}$ ever stops before $\mathcal{A}$ stops, it outputs a correct answer. The claim follows. □ □

There are two options for handling the dynamic protocol version. We could have defined a dynamic version of the general hidden matrix problem and proven a similar security result for that problem. The better approach is to observe that due to compression, we can start with a static problem with some large number of matrices and use it to simulate the general hidden matrix problem for fewer matrices. We leave the details to the interested reader.

More pressing is the fact that the general hidden matrix problem is just the construction from Figure 2 without the hash function, and the security goal, which can be considered a one-way variant of the protocol security goal. In that regard, the above theorem does not say much about the security of the cryptographic construction. We still need to get a better idea of the hardness of the general hidden matrix problem and the gap variation.

One can consider the gap general hidden matrix problem to indicate that the computational and decisional versions of the problem are not equivalent, that is, the computational general hidden matrix problem cannot be reduced to the corresponding decisional problem. Due to the problem structure, finding a non-trivial input to the gap oracle looks like solving a different instance of the same underlying problem. It therefore seems plausible that the gap oracle is of little help, and the gap variant is as hard as the ordinary general hidden matrix problem.

We now consider the selective variant, where the challenge set must be specified before reveal queries are issued. Since the selective variant only removes power from the adversary it can clearly not be easier than the ordinary problem. Left to determine is if the selective variant is harder.

There is a trivial reduction from the ordinary problem to the selective variant; simply by guessing the target J at the start and aborting if the guess turns out to be incorrect. However, since the number of possible guesses is approximately 2^n, this reduction is only effective for very small n. To some extent the weak reduction can be compensated by choosing a larger bilinear group structure, but this does not work very well if n is large. This weak reduction does, however, suggest that if using the reveal oracle before choosing the target confers some advantage, it is not of substantial amount. In other words, even though we do not have an effective reduction, it seems plausible that the selective variant is not much harder than the ordinary problem.

It follows that we want to better understand the hardness of the selective variant. To do that, we return to the construction from Figure 2 and make the basic observation that given two universal tokens $t_1 = (\mathbf{T}_1^{no}, \mathbf{T}_1^{yes})$ and $t_2 = (\mathbf{T}_2^{no}, \mathbf{T}_2^{yes})$ we can reconstruct the product $\mathbf{S}_3 = \mathbf{T}_1^{yes}\mathbf{T}_2^{yes}$ from the three products $\mathbf{S}_0 = \mathbf{T}_1^{yes}\mathbf{T}_2^{no}$, $\mathbf{S}_1 = \mathbf{T}_1^{no}\mathbf{T}_2^{no}$ and

$\mathbf{S}_2 = \mathbf{T}_1^{no}\mathbf{T}_2^{yes}$ as $\mathbf{S}_3 = \mathbf{S}_0\mathbf{S}_1^{-1}\mathbf{S}_2$. If this translates to a collection of personal tokens it poses a major risk to the security guarantee of the proposed protocol, and will result in an adversary with advantage 1. Assume that user one has access to personal yes-token $\mathbf{T}_{\mathsf{id}_1,1}^{yes}$, while user two has knowledge of $\mathbf{T}_{\mathsf{id}_2,2}^{yes}$. Then user 1 can generate a variant of $\mathbf{S}_0$, while user 2 can generate a variant of $\mathbf{S}_2$. Both users can generate versions of $\mathbf{S}_1$. It is, therefore, crucial to determine if the personal $\mathbf{S}$-variants are such that the statement about $\mathbf{S}_3$ still holds. This gives rise to the *hidden matrix* problem.

Definition 3. Let $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$ be cyclic groups of order q, let $e : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_3$ be a bilinear pairing and let $[\cdot]_i$ represent the embedding of a matrix in $\mathcal{G}_i$. We say that the *hidden matrix problem (HMP)* with respect to this structure holds if for all PPT adversaries $\mathcal{A}$, given $[\mathbf{L}]_1$, $[\mathbf{R}]_2$, $[\mathbf{L}\mathbf{A}^{-1}]_1$, $[\mathbf{B}^{-1}\mathbf{R}]_2$, $\mathbf{S}_0$, $\mathbf{S}_1$, $\mathbf{A}\mathbf{S}_1\mathbf{B}$ and $\mathbf{A}\mathbf{S}_2\mathbf{B}$, the probability that $\mathcal{A}$ can compute $[\mathbf{L}\mathbf{S}_0\mathbf{S}_1^{-1}\mathbf{S}_2\mathbf{R}]_3$ is negligible, where $\mathbf{A}, \mathbf{B}, \mathbf{S}_0, \mathbf{S}_1, \mathbf{S}_2 \leftarrow\!\!\$\ \mathbb{Z}_q^{k\times k}$, $\mathbf{L} \leftarrow\!\!\$\ \mathbb{Z}_q^{l\times k}$ and $\mathbf{R} \leftarrow\!\!\$\ \mathbb{Z}_q^{k\times m}$, all of which have maximal rank.

In essence the hidden matrix problem asks whether or not it is hard to extract a matrix, $\mathbf{S}_2$ from a matrix product $\mathbf{A}\mathbf{S}_2\mathbf{B}$ when neither of the hiding matrices $\mathbf{A}$ and $\mathbf{B}$ are know directly, or alternatively whether there might be ways to combine the matrices such that the hiding matrices cancel to provide the desired product.

We discuss the hardness of the hidden matrix problem further in Appendix A. The result of the argument is that the hidden matrix problem is as hard, but no harder than the discrete logarithm problem.

Finally, we show that the selective variant of the general hidden matrix problem reduces to the hidden matrix problem.

Theorem 3. *Let $\mathcal{A}$ be a selective adversary against the general hidden matrix problem with advantage ϵ. Then there exists an adversary $\mathcal{B}$ against the hidden matrix problem with advantage ϵ/n^2, with essentially the same runtime as $\mathcal{A}$.*

Proof. We first argue that we only need two distinct tokens, which means that we only need to guess the two token positions that the adversary will not simultaneously reveal (which accounts for the $1/n^2$ loss in advantage). Then we argue that we only need two users, where one has the first yes-token and the other has the second yes-token, but neither have both. Finally, we argue that for two tokens and two users, we can trivially embed the hidden matrix problem into the stated problem.

First, suppose that $n = 2$. We need to simulate more tokens, with the given tokens placed in position i' and i''. Assume without loss of generality that $i' < i''$. We want to simulate some tokens before token number i', some between token number i' and i'' and some after token number i''. The simulated matrices $\mathbf{T}_i^{f(i,J)}, i \in \{1, 2, \cdots, n'\} \setminus \{i', i''\}$ are sampled such that they cancel out between real tokens. That is, we require that

$$\prod_{i=1}^{i'-1} \mathbf{T}_i^{f(i,J)} = \prod_{i=i'+1}^{i''-1} \mathbf{T}_i^{f(i,J)} = \prod_{i=i''+1}^{n'} \mathbf{T}_i^{f(i,J)} = \mathbf{I}.$$

To create personal tokens, we sample appropriate matrices $\mathbf{A}_i$ for each user. The matrices are multiplied into the corresponding token matrices to create simulated personal tokens. This is done both for real and simulated tokens, meaning that the real tokens will have a double buffer of matrices around the universal matrix. The cancellation requirement of the simulated universal tokens is not visible to the adversary because of the matrices $\mathbf{A}_i$ that do not cancel completely without knowledge of the real tokens.

It follows that we may restrict attention to the two-token case, $n = 2$. In this case some users have the first yes-token, while others have the second yes-token. Any two users that have the first yes-token differ only in the matrices $\mathbf{A}_i$ used to generate their personal tokens and secret keys. It is clear that we can re-randomize key material by sampling

$\tilde{\mathbf{A}}_0$, $\tilde{\mathbf{A}}_1$ and $\tilde{\mathbf{A}}_2$ and multiplying them into the given tokens and representations, thereby simulating further users that have the first yes-token. Likewise we can simulate users that have the second yes-token by starting with one user with the second yes-token and selecting random matrices to re-randomize. In other words, we may assume that $u = 2$.

We now show that when $n = 2$ and $u = 2$, we can embed a hidden matrix problem instance. That is we can use an adversary against token product to create an adversary against the hidden matrix problem. Let $\mathcal{A}$ be an adversary that aims to compute the stated token product.

Given a hidden matrix instance with matrices $[\mathbf{L}]_1$, $[\mathbf{R}]_2$, $[\mathbf{L}\mathbf{A}^{-1}]_1$, $[\mathbf{B}^{-1}\mathbf{R}]_2$, $\mathbf{S}_0$, $\mathbf{S}_1$, $\mathbf{A}\mathbf{S}_1\mathbf{B}$ and $\mathbf{A}\mathbf{S}_2\mathbf{B}$.

Choose random matrices $\mathbf{T}^{no}_{\mathsf{id}_1,1}$ and $\mathbf{T}^{no}_{\mathsf{id}_2,2}$, and compute

$$\begin{aligned} \mathbf{T}^{yes}_{\mathsf{id}_1,1} &= \mathbf{S}_0(\mathbf{T}^{no}_{\mathsf{id}_1,2})^{-1} & \mathbf{T}^{no}_{\mathsf{id}_2,1} &= \mathbf{A}\mathbf{S}_1\mathbf{B}(\mathbf{T}^{no}_{\mathsf{id}_2,2})^{-1} \\ \mathbf{T}^{no}_{\mathsf{id}_1,2} &= (\mathbf{T}^{no}_{\mathsf{id}_1,1})^{-1}\mathbf{S}_1 & \mathbf{T}^{yes}_{\mathsf{id}_2,2} &= (\mathbf{T}^{no}_{\mathsf{id}_2,1})^{-1}\mathbf{A}\mathbf{S}_2\mathbf{B} \end{aligned}.$$

Give the personal tokens to the adversary $\mathcal{A}$.

The idea is to let $\mathbf{A}_{\mathsf{id}_1,0} = \mathbf{I} = \mathbf{A}_{\mathsf{id}_1,2}$, $\mathbf{A}_{\mathsf{id}_2,0} = \mathbf{A}$ and $\mathbf{A}_{\mathsf{id}_2,2} = \mathbf{B}^{-1}$, while the matrices $\mathbf{A}_{\mathsf{id}_1,1}$ and $\mathbf{A}_{\mathsf{id}_2,1}$ give us the needed two degrees of freedom.

We first observe that the personal no-tokens are consistent. Meaning that both users will will generate the same value up to matrix multiplication.

$$\mathbf{A}^{-1}\mathbf{T}^{no}_{\mathsf{id}_2,1}\mathbf{T}^{no}_{\mathsf{id}_2,2}\mathbf{B}^{-1} = \mathbf{A}^{-1}\mathbf{A}\mathbf{S}_1\mathbf{B}(\mathbf{T}^{no}_{\mathsf{id}_2,2})^{-1}\mathbf{T}^{no}_{\mathsf{id}_2,2}\mathbf{B}^{-1} = \mathbf{S}_1 = \mathbf{T}^{no}_{\mathsf{id}_1,1}\mathbf{T}^{no}_{\mathsf{id}_1,2}.$$

Furthermore,

$$\mathbf{S}_0 = \mathbf{T}^{yes}_{\mathsf{id}_1,1}\mathbf{T}^{no}_{\mathsf{id}_1,2} \qquad \mathbf{S}_1 = \mathbf{T}^{no}_{\mathsf{id}_1,1}\mathbf{T}^{no}_{\mathsf{id}_1,2} \qquad \mathbf{A}\mathbf{S}_2\mathbf{B} = \mathbf{T}^{no}_{\mathsf{id}_2,1}\mathbf{T}^{yes}_{\mathsf{id}_2,2}$$

If $\mathcal{A}$ output the correct answer $[\mathbf{L}\mathbf{T}^{yes}_1\mathbf{T}^{yes}_2\mathbf{R}]_3$ then this is also the correct answer for the hidden matrix problem since:

$$\mathbf{T}^{yes}_1\mathbf{T}^{yes}_2 = \mathbf{T}^{yes}_{\mathsf{id}_1,1}\mathbf{T}^{no}_{\mathsf{id}_1,2}(\mathbf{T}^{no}_{\mathsf{id}_1,1}\mathbf{T}^{no}_{\mathsf{id}_1,2})^{-1}\mathbf{A}^{-1}\mathbf{T}^{no}_{\mathsf{id}_2,1}\mathbf{T}^{yes}_{\mathsf{id}_2,2}\mathbf{B}^{-1} = \mathbf{S}_0\mathbf{S}_1^{-1}\mathbf{S}_2$$

as desired.

We have therefore perfectly simulated the distribution of tokens with $n = 2$ and $u = 2$, and the correct answer for $\mathcal{A}$ equals the correct answer for $\mathcal{B}$, which completes the argument. □ □

To summarize, Theorem 2 shows that the security of the construction in Figure 2 follows from the gap variant of the general hidden matrix problem. We have argued that the gap oracle does not help the adversary, so if the general hidden matrix problem is hard, the construction should be secure. Further, we have argued that the general hidden matrix problem should not be much easier than the selective variant. Theorem 3 shows that the selective variant is hard if a simpler problem, the hidden matrix problem (Definition 3), is hard. Finally, we have argued (in Appendix A) that the hidden matrix problem is hard.

In other words, we conjecture that the construction in Figure 2 is secure.

Remark 3. Since it is easy to define a gap variant of the hidden matrix problem, and this variant does not seem to introduce any additional problems with the analysis in Appendix A, it is natural to ask if Theorem 3 could be generalized to cover a reduction from a gap variant of the hidden matrix problem to the gap variant of the selective general hidden matrix problem. This seems to be difficult, since we cannot assume that the two tokens are adjacent, which means that it would be hard to adapt any general hidden matrix gap query to a hidden matrix gap query.

References

[ACDT20] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 248–277, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-56784-2_9.

[BBR+23] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. RFC 9420, July 2023. URL: https://www.rfc-editor.org/info/rfc9420, doi:10.17487/RFC9420.

[BC94] Amos Beimel and Benny Chor. Interaction in key distribution schemes (extended abstract). In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 444–455, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany. doi:10.1007/3-540-48329-2_38.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 258–275, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany. doi:10.1007/11535218_16.

[CCG+18] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1802–1819, Toronto, ON, Canada, October 15–19, 2018. ACM Press. doi:10.1145/3243734.3243747.

[CKS08] David Cash, Eike Kiltz, and Victor Shoup. The twin diffie-hellman problem and applications. In Nigel Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, pages 127–145, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. doi:10.1109/TIT.1976.1055638.

[DLL22] Li Duan, Yong Li, and Lijun Liao. Flexible group non-interactive key exchange in the standard model. In Peter Y.A. Ryan and Cristian Toma, editors, *Innovative Security Solutions for Information Technology and Communications*, pages 210–227, Cham, 2022. Springer International Publishing.

[EHK+13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for Diffie-Hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 129–147, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-40084-1_8.

[FHKP13] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. Non-interactive key exchange. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *Public-Key Cryptography – PKC 2013*, pages 254–271, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[FN94] Amos Fiat and Moni Naor. Broadcast encryption. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 480–491, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany. doi:10.1007/3-540-48329-2_40.

[GBG10] M. Choudary Gorantla, Colin Boyd, and Juan Manuel González Nieto. Attribute-based authenticated key exchange. In Ron Steinfeld and Philip Hawkes, editors, *ACISP 10: 15th Australasian Conference on Information Security and Privacy*, volume 6168 of *Lecture Notes in Computer Science*, pages 300–317, Sydney, NSW, Australia, July 5–7, 2010. Springer, Heidelberg, Germany.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, page 89–98, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1180405.1180418.

[KKL+16] Vladimir Kolesnikov, Hugo Krawczyk, Yehuda Lindell, Alex J. Malozemoff, and Tal Rabin. Attribute-based key exchange with general policies. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1451–1463, Vienna, Austria, October 24–28, 2016. ACM Press. doi:10.1145/2976749.2978359.

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023*, pages 407–441, Singapore, 2023. Springer Nature Singapore.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, pages 457–473, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[TZL17] Fei Tank, Rui Zhang, and Hongda Li. Attribute-based non-interactive key exchange. In *Science China Information Science*. Springer, 2017.

A On the hidden matrix problem

It is not obvious that the hidden matrix problem is hard. In the following, we argue informally. First, we simplify the hidden matrix problem, resulting in a problem that intuitively seems to be hard. Then we sketch an argument for hardness in the generic bilinear group model.

Problem restatement In the hidden matrix problem, the adversary gets access to the following matrices and matrix representations:

$$[\mathbf{L}]_1 \qquad [\mathbf{R}]_2 \qquad [\mathbf{L}\mathbf{A}^{-1}]_1 \qquad [\mathbf{B}^{-1}\mathbf{R}]_2 \qquad \mathbf{S}_0 \qquad \mathbf{S}_1 \qquad \mathbf{A}\mathbf{S}_1\mathbf{B} \qquad \mathbf{A}\mathbf{S}_2\mathbf{B}$$

The goal of the adversary is to compute $[\mathbf{L}\mathbf{S}_0\mathbf{S}_1^{-1}\mathbf{S}_2\mathbf{R}]_3$.

Removing $\mathbf{S}_2$: The first step is remove all traces of the hidden matrix $\mathbf{S}_2$, both in the provided matrices as well as the target product. This is achieved through a re-naming procedure.

We observe that $(\mathbf{AS}_1\mathbf{B})^{-1}\mathbf{AS}_1\mathbf{R} = \mathbf{B}^{-1}\mathbf{R}$. If we set $\mathbf{U}_1 \leftarrow \mathbf{AS}_1\mathbf{B}$ and $\mathbf{U}_2 \leftarrow \mathbf{AS}_2\mathbf{B}$, the original problem can be restated as follows without altering hardness, since this is just a change of variables.

The adversary now gets access to the following matrices and matrix representations:

$$[\mathbf{L}]_1 \qquad [\mathbf{R}]_2 \qquad [\mathbf{LA}^{-1}]_1 \qquad [\mathbf{U}_1^{-1}\mathbf{AS}_1\mathbf{R}]_2 \qquad \mathbf{S}_0 \qquad \mathbf{S}_1 \qquad \mathbf{U}_1 \qquad \mathbf{U}_2$$

The goal of the adversary is to compute $[\mathbf{LS}_0\mathbf{S}_1^{-1}\mathbf{A}^{-1}\mathbf{U}_2\mathbf{U}_1^{-1}\mathbf{AS}_1\mathbf{R}]_3$.

Removing R: Similarly to the previous step we now aim to remove the need for a specific matrix through a change of variables.

Set $\tilde{\mathbf{R}} \leftarrow \mathbf{S}_1\mathbf{R}$ and substitute into the problem, to alter the previous statement accordingly.

The adversary now gets access to the following matrices and matrix representations:

$$[\mathbf{L}]_1 \qquad \mathbf{S}_1^{-1}[\tilde{\mathbf{R}}]_2 \qquad [\mathbf{LA}^{-1}]_1 \qquad \mathbf{U}_1^{-1}[\mathbf{A}\tilde{\mathbf{R}}]_2 \qquad \mathbf{S}_0 \qquad \mathbf{S}_1 \qquad \mathbf{U}_1 \qquad \mathbf{U}_2$$

The goal of the adversary is to compute $[\mathbf{LS}_0\mathbf{S}_1^{-1}\mathbf{A}^{-1}\mathbf{U}_2\mathbf{U}_1^{-1}\mathbf{A}\tilde{\mathbf{R}}]_3$.

Simplifying statement of provided information: Observe again that the adversary is given $\mathbf{S}_1$ and $\mathbf{U}_1$, so it does not actually matter if we give the adversary $\mathbf{S}_1^{-1}[\tilde{\mathbf{R}}]_2$ or just $[\tilde{\mathbf{R}}]_2$. The same applies to $[\mathbf{A}\tilde{\mathbf{R}}]_2$.

The adversary now gets access to the following matrices and matrix representations:

$$[\mathbf{L}]_1 \qquad [\tilde{\mathbf{R}}]_2 \qquad [\mathbf{LA}^{-1}]_1 \qquad [\mathbf{A}\tilde{\mathbf{R}}]_2 \qquad \mathbf{S}_0 \qquad \mathbf{S}_1 \qquad \mathbf{U}_1 \qquad \mathbf{U}_2$$

The goal of the adversary is to compute $[\mathbf{LS}_0\mathbf{S}_1^{-1}\mathbf{A}^{-1}\mathbf{U}_2\mathbf{U}_1^{-1}\mathbf{A}\tilde{\mathbf{R}}]_3$.

More statement simplification: The four matrices given in the clear appear only as specific products in the adversarial goal. We may, therefore, substitute $\mathbf{V}_1$ for $\mathbf{S}_0\mathbf{S}_1^{-1}$ and $\mathbf{V}_2$ for $\mathbf{U}_2\mathbf{U}_1^{-1}$.

The adversary now gets access to the following matrices and matrix representations:

$$[\mathbf{L}]_1 \qquad [\tilde{\mathbf{R}}]_2 \qquad [\mathbf{LA}^{-1}]_1 \qquad [\mathbf{A}\tilde{\mathbf{R}}]_2 \qquad \mathbf{V}_1 \qquad \mathbf{V}_2$$

The goal of the adversary is to compute $[\mathbf{LV}_1\mathbf{A}^{-1}\mathbf{V}_2\mathbf{A}\tilde{\mathbf{R}}]_3$.

Discussion This simpler reformulation suggests that the problem is indeed hard. An adversary would have to force matrices $\mathbf{V}_1$ and $\mathbf{V}_2$ into the middle of a matrix representation without any knowledge or information of $\mathbf{A}$ or the other surrounding matrices. The fact that $\mathbf{A}$ largely remains unknown should follow because discrete logarithms are hard to compute.

Further, depending on the dimensions of $\mathbf{L}$ and $\mathbf{R}$, the adversary may not even have enough information to solve the problem.

In other words, the hidden matrix problem does seem hard.

Remark 4. It is now easy to see that the hidden matrix problem is *not* hard in the one-dimensional case.

Hardness Argument To further justify that the intuition above holds we sketch an argument in the generic bilinear group model.

Given a random bijections to represent cyclic groups $\mathbb{Z}_p$, we introduce variables X_{ij}, Y_{ij}, Z_{ij} and W_{ij} and let the matrices (X_{ij}), (Y_{ij}), (Z_{ij}) and (W_{ij}) represent the group elements $[\mathbf{L}]_1$, $[\tilde{\mathbf{R}}]_2$, $[\mathbf{L}\mathbf{A}^{-1}]_1$ and $[\mathbf{A}\tilde{\mathbf{R}}]_2$, respectively. An ideal I over the resulting polynomial ring is generated by the polynomials in $(X_{ij}) - (Z_{ij})\mathbf{A}$ and $\mathbf{A}(Y_{ij}) - (W_{ij})$, i.e.,

$$I = \langle (X_{ij}) - (Z_{ij})\mathbf{A},\ \mathbf{A}(Y_{ij}) - (W_{ij}) \rangle$$

.

Since $\mathbf{A}$ is random, the odds of any generic bilinear group adversary computing linear polynomials such that a difference lies in the ideal is bounded by the number of queries the adversary can make to the group operation and pairing oracles, in the usual manner.

Let

$$J = \langle (X_{ij})(Y_{ij}) - (Z_{ij})(W_{ij}) \rangle$$

be another ideal. A generic bilinear group adversary can trivially generate elements (polynomials of degree at least 2) in this ideal using the bilinear paring. It is clear that $J \subseteq I$.

We can now compute modulo the ideal J instead of I. Since the matrix $\mathbf{A}$ is unknown and only hidden in the matrix representation, the probability that a generic bilinear group model adversary notices the different modulus should be bounded by the number of queries to the discrete logarithm oracle, in the usual manner.

Finally, we observe that when computing modulo J, the adversary has no information about the matrix $\mathbf{A}$, in which case the matrix is indeed hidden and the hidden matrix problem is trivially hard. *In other words, the hidden matrix problem does seem hard in the generic bilinear group model.*