Two Applications of Multilinear Maps: Group Key Exchange and Witness Encryption

Seiko Arita
Institute of Information Security
Kanagawa, Japan
arita@iisec.ac.jp

Sari Handa
Institute of Information Security
Kanagawa, Japan
mgs125502@iisec.ac.jp

ABSTRACT

Constructing multilinear maps has been long-standing open problem, before recently the first construction based on ideal lattices has been proposed by Garg et al. After this breakthrough, various new cryptographic systems have been proposed. They introduce the concept of level into the encodings, and the system has a function that extracts a deterministic value at only a specific level, and the encodings are unable to downgrade to the lower levels. These properties are useful for cryptography. We study how this graded encoding system be applied to cryptosystems, and we propose two protocols, group key exchange and witness encryption. In our group key exchange, we achieve the communication size and the computation costs per party are both O(1) with respect to the number of parties by piling the encodings of passed parties in one encoding. A witness encryption is a new type cryptosystem using NP-complete problem. The first construction is based on EXACT-COVER problem. We construct it based on another NP-complete Hamilton Cycle problem, and prove its security under the Generic Cyclic Colored Matrix Model.

Categories and Subject Descriptors

E.3 [Data] [**Data Encryption**]: [public key cryptosystems]

General Terms

Theory,Security

Keywords

multilinear maps, group key exchange, witness encryption, hamilton cycle problem

1. INTRODUCTION

Multilinear maps have been desired to be constructed because if it exists it would enable many interesting cryptographic applications. Boneh and Silverberg attempted to construct multilinear maps from abelian varieties, but they concluded such maps should be hard to construct[BS03]. Recently, the first construction of multilinear maps has been described by Garg, Gentry and Halevi based on ideal lattices [GGH13]. After this breakthrough, various new cryptographic systems have been proposed today. The system realizing multilinear maps is called "graded encoding system". The encodings have homomorphic addition and multiplication and they introduce the concept of *level* into the encodings. This concept of level making the encoding system interesting, here we describe two of its interesting properties. First, an encoding is probabilistic, that is, encodings of a same plaintext are different because of included randomness, but at only a specified level, a deterministic value can be extracted from the random encodings of the same plaintext. Second property is that a rerandomized encoding is not allowed to be divided by some other encodings, so an encoding is unable to downgrade to the lower levels. Applications use these properties. For example, in the group key exchange on N parties [GGH13] [CLT13], each party creates a level-$(N-1)$ encoding from its own encoding and others, and they extract a same value as a sheared group key at the level-$(N-1)$. In witness encryption [GGSW], an encryptor and a decryptor generate a same level encoding for each by different way and extracts a same value from their own created encodings. The second property brings one-wayness and means a strong tool for cryptosystems. In the work of attribute based encryption for circuit [GGHSW13], authors use this one-wayness to prevent a back tracking attack and achieved to treat circuits with multi fun-out gates.

Our result. We have studied applications of multilinear maps. In this work we propose two such applications: Group Key Exchange and Witness Encryption. A one-round group key exchange protocol is described in the work [GGH13] [CLT13], in which each party collects encodings of all other parties and multiplies its own encoding and them one-by-one. We design a GKE protocol in which one encoding is communicated on each upflow and downflow, in which parties's secret are piled up gradually. The number of encodings communicated in a session of the GKE protocol per party does not depend on the number of parties. Also its computation cost per party is independent of the number of parties because each party only multiplies a received encoding by its own encoding.

Witness encryption is a new type of cryptosystem that can be achieved using multilinear maps [GGSW]. The witness encryption of [GGSW] uses EXACT-COLVER Problem as NP-complete language. We try to construct based on an-

AsiaPKC'14, June 3, 2014, Kyoto, Japan.

http://dx.doi.org/10.1145/2600694.2600699.

other NP-complete problem, Hamilton Cycle Problem. The two problems have in common with a task of collecting just N element with no duplication, and this situation matches the property of the graded encoding system that extracts a deterministic value only from the specified level-N encodings. The difference is that in the case of Hamilton Cycle Problem, elements are connected by edges and a cycle follows the adjacent edges sequentially. We take in a tool of adjacent matrices for managing edge adjacency and manages non-commutativity of vertices in a cycle by matrix's non-commutativity. We proved the security of our scheme based on our generic cyclic colored matrix model that is a variant of a generic colored matrix model defined by work of indistinguishability obfuscation [GGH13+].

2. PRELIMINARIES

2.1 Approximate Multilinear Maps

Gentry et al. defined their notion of a approximate multilinear maps, which they call *graded encoding schemes* [GGH13]. They view group element in multilinear map schemes as just a convenient mechanism of encoding the exponent: Typical application of bilinear maps use $\alpha \cdot g_i$ as an encoding of the plaintext integer $\alpha \in \mathbb{Z}_p$. In their setting, they retain the concept of a somewhat homomorphic encoding, and have an algebraic ring (or field) R playing the role of the exponent space $\mathbb{Z}_p$.

2.1.1 Definition of Graded Encoding Schemes

DEFINITION 1. *(κ-Graded Encoding System [GGH13]). A κ-Graded Encoding System consists of a ring R and a system of sets $\mathsf{S} = \{S_i^{(\alpha)} \subset \{0,1\}^* : i \in [0,\kappa], \alpha \in R\}$, with the following properties:*
1. For every fixed index $i \in [0,\kappa]$, the sets $\{S_i^{(\alpha)} : \alpha \in R\}$ are disjoint. The set $S_i^{(\alpha)}$ consists of the "level-i encodings of α".
2. There is an associative binary operation "+" and a self-inverse unary operation "-" (on $\{0,1\}^*$) such that for every $\alpha_1, \alpha_2 \in R$, every index $i \leq \kappa$, and every $u_1 \in S_i^{(\alpha_1)}$ and $u_2 \in S_i^{(\alpha_2)}$, it holds that $u_1 + u_2 \in S_i^{(\alpha_1+\alpha_2)}$ and $-u_1 \in S_i^{-\alpha_1}$ where $\alpha_1 + \alpha_2$ and $-\alpha_1$ are addition and negation in R.
3. There is an associative binary operation "$\times$" (on $\{0,1\}^*$) such that for every $\alpha_1, \alpha_2 \in R$, every i_1, i_2 with $i_1 + i_2 \leq \kappa$, and every $u_1 \in S_{i_1}^{(\alpha_1)}$ and $u_2 \in S_{i_2}^{(\alpha_2)}$, it holds that $u_1 \times u_2 \in S_{i_1+i_2}^{(\alpha_1\alpha_2)}$. Here $\alpha_1 \cdot \alpha_2$ is multiplication in R, and $i_1 + i_2$ is integer addition.

The n-graded encoding system of [GGH13] for a ring R includes a system of sets $S = \{S_i^{(\alpha)} \subset \{0,1\}^* : i \in [0,n], \alpha \in R\}$ such that, for every fixed $i \in [0,n]$, the sets $\{S_i^{(\alpha)} : \alpha \in R\}$ are disjoint. The set $S_i^{(\alpha)}$ consists of the "level-i encodings of α".

Instance Generation. The randomized $\mathsf{InstGen}(1^\lambda, 1^n)$ takes as inputs the security parameter λ and integer n. The procedure outputs $(\mathsf{params}, p_{zt})$, where params is a description of an n-graded encoding system, and p_{zt} is a level-n "zero-test parameter".

Ring Sampler. The randomized $\mathsf{samp}(\mathsf{params})$ outputs a "level-zero encoding" a $a \in S_0$, such that the induced distribution on α such that a $a \in S_0^{(\alpha)}$ is statistically uniform.

Encoding. The $\mathsf{enc}(\mathsf{params}, i, a)$ takes $i \in [n]$ and a level-zero encoding $a \in S_0^{(\alpha)}$ for some $\alpha \in R$, and outputs a level-i encoding $u \in S_i^{(\alpha)}$ for the same α.

Re-Randomization. The $\mathsf{reRand}(\mathsf{params}, i, u)$ re-randomizes encodings to the same level i, as long as the initial encoding u is under a given noise bound.

Addition and negation. Given params and two encodings at the same level, $u_1 \in S_i^{(\alpha_1)}$ and $u_2 \in S_i^{(\alpha_2)}$, it holds $\mathsf{add}(\mathsf{params}, u_1, u_2) \in S_i^{(\alpha_1+\alpha_2)}$, and $\mathsf{neg}(\mathsf{params}, u_1) \in S_i^{(-\alpha_1)}$, subject to bounds on the noise.

Multiplication. For $u_1 \in S_{i_1}^{(\alpha_1)}$,$u_2 \in S_{i_2}^{(\alpha_2)}$, such that $i_1 + i_2 \leq n$, we have $\mathsf{mult}(\mathsf{params}, u_1, u_2) \in S_{i_1+i_2}^{(\alpha_1\alpha_2)}$.

Zero-test. The procedure $\mathsf{isZero}(\mathsf{params}, p_{zt}, u)$ outputs 1 if $u \in S_n^{(0)}$ and 0 otherwise. Note that in conjunction with the procedure for subtracting encodings, this gives us an equality test.

Extraction. This procedure extracts a "canonical" and "random" representation of ring elements from their level-n encoding. Namely $\mathsf{ext}(\mathsf{params}, p_{zt}, u)$ outputs (say) $K \in \{0,1\}^\lambda$, such that:
(a) With overwhelming probability over the choice of $\alpha \in R$, for any two $u_1, u_2 \in S_n^{(\alpha)}$, $\mathsf{ext}(\mathsf{params}, p_{zt}, u_1) = \mathsf{ext}(\mathsf{params}, p_{zt}, u_2)$,
(b) The distribution $\{\mathsf{ext}(\mathsf{params}, p_{zt}, u) : \alpha \in R, u \in S_n^{(\alpha)}\}$ is statistically uniform over $\{0,1\}^\lambda$.

2.1.2 Graded Decisional Diffie-Hellman Problem

Garg et al. [GGH13] and Coron et al. [CLT13] define Graded DDH Problem (GDDHProblem), as following process:

1. $(\mathsf{params}, p_{zt}) \leftarrow \mathsf{InstGen}(1^\lambda, 1^\kappa)$
2. Choose $a_j \leftarrow \mathsf{samp}(\mathsf{params})$ for all $1 \leq j \leq \kappa + 1$
3. Set $u_j \leftarrow \mathsf{reRand}(\mathsf{params}, 1, \mathsf{enc}(\mathsf{params}, 1, a_j))$ for all $1 \leq j \leq \kappa + 1$
4. Choose $b \leftarrow \mathsf{samp}(\mathsf{params})$
5. Set $\tilde{u} = a_{\kappa+1} \times \Pi_{i=1}^{\kappa} u_i$
6. Set $\hat{u} = b \times \Pi_{i=1}^{\kappa} u_i$

The GDDH distinguisher is given as input the $\kappa+1$ level-one encodings u_j and either $\tilde{u}$ (encoding the right product) or $\hat{u}$ (encoding a random product), and must decide which is the case.

Graded Decisional Diffie-Hellman Assumption (GDDH Assumption) The Graded Decisional Diffie-Hellman Assumption is that the advantage of any efficient adversary is negligible in the security parameter against Graded Decisional Diffie-Hellman Problem.

3. GROUP KEY EXCHANGE USING MULTILINEAR MAPS

The graded encoding system has a function that extracts a deterministic value associated with a plaintext from a specified level encodings and the level is specified at the instance generation. This functionality fits with a group key exchange (GKE) because the shared key should be generated from secrets of all parties. We construct a GKE protocol which communication type is (N-1)-round 2-way of upflow and downflow. The construction is simple, each party multiplies a received encoding by own encoding and sends the

Table 1: Comparison of performance with respect to the number of parties (per party)

Scheme	Comm.	Comp.
One-round using MM [GGH13]	$O(N)$	$O(N)$
upflow/downflow no MM [STW96]	$O(N)$	$O(N)$
upflow/downflow using MM (ours)	$O(1)$	$O(1)$

result encoding to the next party. In the communicated encoding, secrets of passed parties are piled up, so the message complexity per party does not depend on the number of parties. Also, the computation cost for each party is $O(1)$.

In Table 1, we compare our GKE scheme with two other schemes in terms of their performance with respect to the number of parties, 'N' means the number of parties, 'Comm.' is communication size and 'Comp.' is computation cost. The first scheme is one-round GKE using Multilinear Maps in [GGH13]. In this scheme, communicated encodings are not product, so the number of encodings per party is $O(N)$ and the cost one party multiplies received encodings is $O(N)$. The second scheme [STW96] is upflow/downflow type like ours but it does not use Multilinear Maps. The number of communicated group elements per party is $O(N)$ and the computation cost per party is also $O(N)$.

In our protocol, we design the shared group key as output value of pseudo-random function which seed is a level-$(N-1)$ encoding generated by above process and which input value is a session ID. We use graded encoding system for the session ID to achieve theoretically that the conputation cost and communication size per party are $O(1)$. So we make two instances of graded encoding system for level-$(N-1)$ and level-N.

We prove our protocol under GDDH assumption using security model proposed by Bresson et al [BCPQ].

3.1 BCPQ Model : Security Model for GKE

BCPQ Model is a formal model for Group Key Exchange protocol (GKE) proposed by Bresson, Chevassut, Pointcheval, and Quisquater [BCPQ]. In their model, the adversary $\mathcal{A}$ controls all communication between player instances and asks an instance to release session key or long-lived key. A set ID of n players in protocols P is fixed. A player $U_i \in ID$ can have many instances called oracles, involved in distinct concurrent execution of P. An instance s of player U_i is denoted as oracle Π_i^s.

SessionIDS. Session id for oracle Π_i^s is defined as $SIDS(\Pi_i^s) = \{SIDS_{ij} : j \in ID\}$ where $SIDS_{ij}$ is the concatenation of all flows that Π_i^s exchanges with Π_j^t in an execution of P. Adversary $\mathcal{A}$ listens on the wire and can constructs SIDS.

Oracle Queries. Adversaries can send following queries to oracles:

·$\mathsf{Send}(\Pi_U^s, m)$:Adversary $\mathcal{A}$ gets the response which Π_U^s have generated in processing incoming-message m.

·$\mathsf{Reveal}(\Pi_U^s)$:This query forces Π_U^s to release its session key.

·$\mathsf{Corrupt}(U)$:Adversary $\mathcal{A}$ gets long-lived key LL_U of U but does not get the internal data of instance of U.

·$\mathsf{Test}(\Pi_U^s)$:$\mathcal{A}$ gets back session key or random string from Π_U^s.

Definition 2 (Correctness). *A GKE protocol is* correct *if for any operation execution between the oracles* $\Pi_{U_1}^s, \cdots, \Pi_{U_N}^s$ *with the same session ids* $\mathrm{sid} = \mathrm{SIDS}(\Pi_i^s)$ *all oracles accept with the same session group key.*

Definition 3 (AKE Security). *Protocol P is called* AKE-secure *if advantage of any efficient $\mathcal{A}$ in the following game $Game^{AKE}(\mathcal{A})$ is negligible.*

$Game^{AKE}$: Adversary $\mathcal{A}$ can ask queries except Test many times, and once, $\mathcal{A}$ sends a Test-query to a *fresh* oracle (a oracle is fresh if nobody has been asked for $\mathsf{Corrupt}$-query at that moment, and no Reveal-query is asked to the oracle or its partners). $\mathcal{A}$ gets back its session key or a random string. The adversary wins if she correctly guesses the bit b used in the above game, and the advantage is probability of win minus 1/2, taken over all bit tosses.

3.2 Our Construction

Our scheme consists of PPT algorithms Setup, Upflow, Downflow and KeyGen, and parties are indexed Party_1 to Party_N, and Party_i is connected to Party_{i+1} $(1 \le i \le N-1)$. This GKE protocol first executes the Setup algorithm and the resulting public information should be shared among parties. Party_1 executes Upflow_1 without input value. Then, Party_i executes Upflow_i, receiving the outputs $(c_{ur}, d_{ur}, \sigma_{ur})$ of its precedent Upflow_{i-1} as inputs, sequentaily for $i = 2$ to N. Then, Party_N executes $\mathrm{Downfow}_N$ without input value and Party_i executes $\mathrm{Downflow}_i$, receiving the outputs $(c_{dr}, d_{dr}, \sigma_{dr})$ of the precedent $\mathrm{Downflow}_{i+1}$ as inputs, also sequentaily for $i = (N-1)$ to 1. In the Downflow sequence, each $\mathrm{Downflow}_i$ invokes the algorithm Keygen to compute the shared session-key ξ among the parties.

Building blocks are multilinear maps MM, EUF-CMA secure signature scheme Σ and pseudo random function H_{seed}. In the following algorithm, $a \cdot b$ denotes $\mathsf{mult}(\mathsf{params}, a, b)$.

Setup. The algorithm Setup takes security parameter λ and number of parties N as input, makes two sets of MM parameter by instance generation specifying level $N-1$ and N, respectively. Then it generates N pairs of signing key sk_i and verification key vk_i by key generation of Σ, and sets signing keys to each parties. The algorithm outputs two sets of parameter of MM and all verification keys $\{vk_i\}$ as public parameter.

$\mathsf{Setup}(1^\lambda, 1^N)$
  $(\mathsf{params}_1, p_{zt1}) \leftarrow \mathsf{MM.InstGen}(1^\lambda, 1^{N-1})$
  $(\mathsf{params}_2, p_{zt2}) \leftarrow \mathsf{MM.InstGen}(1^\lambda, 1^N)$
  for $i = 1$ to N
    $(vk_i, sk_i) \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$
    Sets sk_i to U_i
  return $(\mathsf{params}_1, p_{zt1}, \mathsf{params}_2, p_{zt2}, vk_i(1 \le i \le N))$

Upflow. The algorithm Upflow takes encoding c_{ur}, d_{ur} and signature σ_{ur} as input and verifies them on sender's verification key vk_{i-1}. Then it creates a secret level-0 encoding a_i and upgrades to level-1 and rerandomizes that using param_1. The algorithm creates one more level-1 encoding d_1 for sessionID using param_2. Then, multiplies them by received encoding and outputs the two encodings with its signature σ_{us}.

$\mathsf{Upflow_i}(c_{ur}, d_{ur}, \sigma_{ur})$
  if $i \neq 1$ and $\Sigma.\mathsf{verify}(vk_{i-1}, c_{ur} \| d_{ur}, \sigma_{ur})$=false then abort
  $a_i \leftarrow \mathsf{MM.samp}(\mathsf{params}_1)$

$c_i \leftarrow \mathsf{MM.reRand}(\mathsf{params}_1, 1, \mathsf{MM.enc}(\mathsf{params}_1, 1, a_i))$
$d_i \leftarrow \mathsf{MM.reRand}(\mathsf{params}_2, 1, \mathsf{MM.enc}(\mathsf{params}_2, 1$
$, \mathsf{MM.samp}(\mathsf{params}_2)))$
if $i \neq N$ then
  if $i = 1$ then $c_{us} := c_i \quad d_{us} := d_i$
  else $c_{us} := c_{ur} \cdot c_i \quad d_{us} := d_{ur} \cdot d_i$
  $\sigma_{us} \leftarrow \Sigma.\mathsf{sign}(sk_i, c_{us} \| d_{us})$
  return $(c_{us}, d_{us}, \sigma_{us})$

Downflow. The algorithm Downflow takes encodings c_{dr},d_{dr} and signature σ_{dr} as input and verifies them on sender's verification key vk_{i+1}. Then it generates a session key ξ by key generation algorithm, and multiplies own two encoding c_i and d_i by received encoding c_{dr} and d_{dr} respectively, and outputs the result encodings with its signature σ_{ds}.

$\mathsf{Downflow_i}(c_{dr}, d_{dr}, \sigma_{dr})$
if $i \neq N$ and $\Sigma.\mathsf{verify}(vk_{i+1}, c_{dr} \| d_{dr}, \sigma_{dr})$=false then abort
if $i \neq 1$ then
  if $i = N$ then $c_{ds} := c_i \quad d_{ds} := d_i$
  else $c_{ds} := c_{dr} \cdot c_i \quad d_{ds} := d_{dr} \cdot d_i$
  $\sigma_{ds} \leftarrow \Sigma.\mathsf{Sign}(sk_i, c_{ds} \| d_{ds})$
$\xi \leftarrow \mathsf{KeyGen}(a_i, c_{ur}, c_{dr}, d_i, d_{ur}, d_{dr})$
return $(c_{ds}, d_{ds}, \sigma_{ds})$ as message and ξ as local output.

KeyGen. The algorithm KeyGen takes encodings a_i, c_{ur}, c_{dr} $, d_i,\ d_{ur}, d_{dr}$ as input, multiplies a_i, c_{ur}, c_{dr} for seed of H and d_i, d_{ur}, d_{dr} for sessionID. MM.ext extracts a deterministic value from level-$(N-1)$ encoding c_i', and KeyGen algorithm sets it to seed of H. A deterministic value from level-N encoding d_i' is sessionID and input to H.

$\mathsf{KeyGen_i}(a_i, c_{ur}, c_{dr}, d_i, d_{ur}, d_{dr})$
$c_i' := a_i \cdot c_{ur} \cdot c_{dr} \quad d_i' := d_i \cdot d_{ur} \cdot d_{dr}$
$\delta \leftarrow \mathsf{MM.ext}(\mathsf{params}_1, p_{zt1}, c_i')$
$sessionID \leftarrow \mathsf{MM.ext}(\mathsf{params}_2, p_{zt2}, d_i')$
return $\xi \leftarrow \mathsf{H}_\delta(sessionID)$

Correctness. We show all parties share a same session key string. Considering output string from H in KeyGen algorithm, seed δ of H is the product of own private level-0 encoding and level-1 encodings of all other parties. MM.ext outputs the same string that is a deterministic value of product of plaintexts a_i of all parties. SessionID is product of level-1 encodings d_i of all parties, so MM.ext outputs the same value for each parties.

3.3 Proof of Security

THEOREM 1. *Our protocol is AKE secure under the GDDH Assumption (see Section A.2), and the assumptions that the signature scheme Σ is EUF-CMA and the function H is secure PSF.*

Proof. We prove the security of our protocol based on BCPQ security model.

Game0: Original Security game for our protocol
–Send(null,"*start*")
Follow the instruction of the Setup algorithm to compute and return $(\mathsf{params}_1, p_{zt1}, \mathsf{params}_2, p_{zt2}, vk_i (1 \leq i \leq N))$.

–Send(Π_i^s, $\mathsf{Upflow}(c_{ur}, d_{ur}, \sigma_{ur})$)
Follow the instruction of the $\mathsf{Upflow_i}$ algorithm to compute and return $(c_{us}, d_{us}, \sigma_{us})$.

–Send(Π_i^s, $\mathsf{Downflow}(c_{dr}, d_{dr}, \sigma_{dr})$)
Follow the instruction of the $\mathsf{Downflow_i}$ algorithm to compute and return $(c_{ds}, d_{ds}, \sigma_{ds})$ and local output ξ.

–Reveal(Π_i^s): return ξ
–Corrupt(U_i): return sk_i

–Test(Π_i^s)
$b \xleftarrow{\$} \{0,1\}$
if $b = 0$ then return ξ else return $r \xleftarrow{\$} \{0,1\}^\lambda$

Then adversary $\mathcal{A}$ outputs guess of b as $\hat{b}$.

We define S_i to be the event that $b = \hat{b}$ in Game$i$.

Game1: We make a transition into Game1: Abort if message is fabricated.

–Send(Π_i^s, $\mathsf{Upflow}(c_{ur}, d_{ur}, \sigma)$)
if $c_{ur} \| d_{ur} \notin m'$ and $\Sigma.\mathsf{verify}(vk_{i-1}, c_{ur} \| d_{ur}, \sigma_{ur})$=true
  then abort
if $i \neq 1$ and $\Sigma.\mathsf{verify}(vk_{i-1}, c_{ur} \| d_{ur}, \sigma_{ur})$=false then abort
$m' := m' \| c_{ur} \| d_{ur}$
// following is same as Game0

–Send(Π_i^s, $\mathsf{Downflow}(c_{dr}, d_{dr}, \sigma)$)
if $c_{dr} \| d_{dr} \notin m'$ and $\Sigma.\mathsf{verify}(vk_{i+1}, c_{dr} \| d_{dr}, \sigma_{dr})$=true
  then abort
if $i \neq N$ and $\Sigma.\mathsf{verify}(vk_{i+1}, c_{dr} \| d_{dr}, \sigma_{dr})$=false then abort
$m' := m' \| c_{dr} \| d_{dr}$
// following is same as Game0

CLAIM 1. *If the signature scheme Σ is EUF-CMA secure, then* $|\Pr[S_0] - \Pr[S_1]| \leq negl(\lambda)$.

Sketch of Proof. Let c and d be received encodings, m' be a set of messages that had been verified, vk be a verification key and σ be a signature. We define failure event F that "$(c,d) \notin m'$ and $\Sigma.\mathsf{verify}(vk, c\|d, \sigma) = \text{true}$". Then, $S_0 \wedge \neg F \Leftrightarrow S_1 \wedge \neg F$. Let $\mathcal{A}$ be an adversary that can get an advantage in distinguishing between Game0 and Game1. We construct a forger $\mathcal{F}$ from $\mathcal{A}$.

Forger $\mathcal{F}$. $\mathcal{F}$ guesses a party j whose signature $\mathcal{A}$ will forge. In Setup algorithm, $\mathcal{F}$ sets input vk to vk_j of party j. $\mathcal{A}$ sends Send-query to Π_i^s, if i=j and event F then $\mathcal{F}$ outputs the $((c,d),\sigma)$. $\mathcal{F}$ calls signing oracle to get signature of j.

The probability of $\mathcal{F}$ successes in above game is $1/\mathrm{N} \cdot \Pr[F]$ and signature scheme Σ is EUF-CMA, so $\Pr[F] \leq negl(\lambda)$. From Difference Lemma defined by Shoup in Sequence of Games [Shoup], $|\Pr[S_0] - \Pr[S_1]| \leq \Pr[F]$.
$\therefore |\Pr[S_0] - \Pr[S_1]| \leq negl(\lambda)$. □

Game2: We make a transition into Game2. In this game, seed of H is changed into random encoding which is formed by GDDH Problem instance.

–Test(Π_i^s)

$b \xleftarrow{\$} \{0,1\}$
if $b = 0$ then
$\boxed{a' \leftarrow \mathsf{MM.samp}(\mathsf{params}_1)}$
$c_i' := a' \cdot c_{ur} \cdot c_{dr} \quad d_i' := d_i \cdot d_{ur} \cdot d_{dr}$
$\delta \leftarrow \mathsf{MM.ext}(\mathsf{params}_1, p_{zt1}, c_i')$
$sessionID \leftarrow \mathsf{MM.ext}(\mathsf{params}_2, p_{zt2}, d_i')$
$\xi \leftarrow \mathrm{H}_\delta(sessionID)$
return ξ
else return $r \xleftarrow{\$} \{0,1\}^\lambda$

CLAIM 2. *Under GDDH the assumption,* $|\Pr[\mathrm{S}_1] - \Pr[\mathrm{S}_2]| \leq negl(\lambda)$.

Proof. We construct a distinguisher $\mathcal{D}$ of GDDH Problem from adversary $\mathcal{A}$ against our protocol. $\mathcal{D}$ guesses a party whom $\mathcal{A}$ will ask Test-query. Its instance and partner instances are denoted by Π_i^{s*}. We define a event Guess that is "All Π_i^{s*} are correct". If input of $\mathcal{D}$ is $\tilde{u}$ then the view of $\mathcal{A}$ in $\mathcal{D}$ is view of $\mathcal{A}$ in Game1, and if input of $\mathcal{D}$ is $\hat{u}$ then that is view of $\mathcal{A}$ in Game2.
Algorithm $\mathcal{D}(u_j(1 \leq j \leq N), T = \tilde{u}$ or $\hat{u})$
$s_i^* \xleftarrow{\$}$[1,S] //S is upper bound of numbers of instances $\mathcal{A}$ creates.

–Send(Π_i^s, $\mathsf{Upflow}(c_{ur}, d_{ur}, \sigma)$)
$\boxed{\text{if } s = s_i^* \text{ then}}$
if $c_{ur} \| d_{ur} \notin m'$ and $\Sigma.\mathsf{verify}(vk_{i-1}, c_{ur} \| d_{ur}, \sigma_{ur})$=true then abort
if $i \neq 1$ and $\Sigma.\mathsf{verify}(vk_{i-1}, c_{ur} \| d_{ur}, \sigma_{ur})$=false then abort
$m' := m' \| c_{ur} \| d_{ur}$
$\boxed{c_i \leftarrow u_i}$
$d_i \leftarrow \mathsf{MM.reRand}(\mathsf{params}_2, 1, \mathsf{MM.enc}(\mathsf{params}_2, 1, \mathsf{MM.samp}(\mathsf{params}_2)))$
if $i \neq N$ then
if $i = 1$ then $c_{us} := c_i \quad d_{us} := d_i$
else $c_{us} := c_{ur} \cdot c_i \quad d_{us} := d_{ur} \cdot d_i$
$\sigma_{us} \leftarrow \Sigma.\mathsf{Sign}(sk_i, c_{us} \| d_{us})$
return $(c_{us}, d_{us}, \sigma_{us})$
$\boxed{\text{else}}$ // $s \neq s_i^*$, following is same as Game1

–Send(Π_i^s, $\mathsf{Downflow}(c_{dr}, d_{dr}, \sigma)$)
$\boxed{\text{if } s = s_i^* \text{ then}}$
if $c_{dr} \| d_{dr} \notin m'$ and $\Sigma.\mathsf{verify}(vk_{i+1}, c_{dr} \| d_{dr}, \sigma_{dr})$=true then abort
if $i \neq N$ and $\Sigma.\mathsf{verify}(vk_{i+1}, c_{dr} \| d_{dr}, \sigma_{dr})$=false then abort
$m' := m' \| c_{dr} \| d_{dr}$
if $i \neq 1$ then
if $i = N$ then $c_{ds} := c_i \quad d_{ds} := d_i$
else $c_{ds} := c_{dr} \cdot c_i \quad d_{ds} := d_{dr} \cdot d_i$
$\sigma_{ds} \leftarrow \Sigma.\mathsf{Sign}(sk_i, c_{ds} \| d_{ds})$
$\boxed{\text{// MM.KeyGen is not called.}}$
return (c_{ds}, σ_{ds})
$\boxed{\text{else}}$ // $s \neq s_i^*$, following is same as Game2

–Reveal(Π_i^s)
$\boxed{\text{if } s = s_i^* \text{ then abort with a random bit}}$
return ξ

–Corrupt(U_i): return sk_i

–Test(Π_i^s)
$\boxed{\text{if } s \neq s_i^* \text{ then abort with a random bit}}$
$b \xleftarrow{\$} \{0,1\}$
if $b = 0$ then
$d_i' := d_i \cdot d_{ur} \cdot d_{dr}$
$\boxed{\delta \leftarrow \mathsf{MM.ext}(\mathsf{params}_1, p_{zt1}, T)}$
$sessionID \leftarrow \mathsf{MM.ext}(\mathsf{params}_2, p_{zt2}, d_i')$
return $\xi \leftarrow \mathrm{H}_\delta(sessionID)$
else return $r \xleftarrow{\$} \{0,1\}^\lambda$

The advantage of D is $\mathrm{Adv}_D = |\Pr[\mathrm{D}(u_j, T)=1 \mid T = \tilde{u}]$ - $\Pr[\mathrm{D}(u_j, T)=1 \mid T = \hat{u}]|$. Now we describe $\Pr[\mathrm{D}(u_j, T)=1 \mid T = \tilde{u}]$ as $\Pr[\mathrm{D}(\tilde{u})=1]$, $\Pr[\mathrm{D}(u_j, T)=1 \mid T = \hat{u}]$ as $\Pr[\mathrm{D}(\hat{u})=1]$.

$$\begin{aligned}\Pr[\mathrm{D}(\tilde{u})=1] &= \Pr[\mathrm{D}(\tilde{u})=1 \wedge \neg \text{ Guess}] + \Pr[\mathrm{D}(\tilde{u})=1 \wedge \text{Guess}] \\ &= 1/2 + \Pr[\text{Guess}]\Pr[\mathrm{D}(\tilde{u})=1 \mid \text{Guess}] \\ &= 1/2 + \Pr[\text{Guess}]\Pr[\mathrm{S}_1]\end{aligned}$$

$$\begin{aligned}\Pr[\mathrm{D}(\hat{u})=1] &= \Pr[\mathrm{D}(\hat{u})=1 \wedge \neg \text{ Guess}] + \Pr[\mathrm{D}(\hat{u})=1 \wedge \text{Guess}] \\ &= 1/2 + \Pr[\text{Guess}]\Pr[\mathrm{D}(\hat{u})=1 \mid \text{Guess}] \\ &= 1/2 + \Pr[\text{Guess}]\Pr[\mathrm{S}_2]\end{aligned}$$

$\mathrm{Adv}_D = |\Pr[\mathrm{D}(\tilde{u})=1] - \Pr[\mathrm{D}(\hat{u})=1]| = \Pr[\text{Guess}] \cdot (\Pr[\mathrm{S}_1] - \Pr[\mathrm{S}_2])$
From GDDH assumption $\mathrm{Adv}_D \leq negl(\lambda)$, and Pr[Guess] is $1/\mathrm{poly}(\lambda)$. Therefore $|\Pr[\mathrm{S}_1] - \Pr[\mathrm{S}_2]| \leq negl(\lambda)$. □

Game3: We make a transition into Game3. In this game, the value of H is changed into a random string.

–Test(Π_i^s)
$b \xleftarrow{\$} \{0,1\}$
if $b = 0$ then return $\boxed{r' \xleftarrow{\$} \{0,1\}^\lambda}$
else return $r \xleftarrow{\$} \{0,1\}^\lambda$

CLAIM 3. *If H is secure then* $|\Pr[\mathrm{S}_2] - \Pr[\mathrm{S}_3]| \leq negl(\lambda)$.

Proof. Immediate from security of pseudo-random function. □

The advantage of $\mathcal{A}$ is clearly negligible in Game3 and from Claim1,Claim2,Claim3, the advantage of $\mathcal{A}$ of real is also negligible. □

4. WITNESS ENCRYPTION USING MULTILINEAR MAPS

4.1 Preliminaries

4.1.1 Witness Encryption

A witness encryption scheme [GGSW] for an NP language L(with corresponding witness relation R) consists of the following two polynomial-time algorithms:

Encryption. The algorithm $\mathsf{Encrypt}(1^\lambda, x, m)$ takes as input a security parameter λ, an unbounded-length string x, and a message $m \in \mathcal{M}$, and outputs a ciphertext CT.

Decryption. The algorithm $\mathsf{Decrypt}(CT, w)$ takes as input a ciphertext CT and an unbounded-length string w, and outputs a message m or the symbol $\perp$.

· Correctness. For any security parameter λ, for any m and for any $x \in L$ s.t. $R(x,w)$ holds, we have that Pr[$\mathsf{Decrypt}(\mathsf{Encrypt}(1^\lambda, x, m), w) = m] = 1 - negl(\lambda)$.
· Soundness Security. For any $x \notin L$, for any PPT adversary $\mathcal{A}$ and messages $m_0, m_1 \in \mathcal{M}$,
| Pr[$\mathcal{A}(\mathsf{Encrypt}(1^\lambda, x, m_0)) = 1$] - Pr[$\mathcal{A}(\mathsf{Encrypt}(1^\lambda, x, m_1)) = 1$] $| \leq negl(\lambda)$.

The Security-Correctness Gap. Correctness requires an algorithm can decrypt if $x \in L$ and it knows w s.t. $R(x,w)$. Soundness Security requires if $x \notin L$ then no PPT algorithm can decrypt. This security does not define on the case when $x \in L$. But since distinguishing $x \in L$ and $x \notin L$ is NP-complete, decryption would be difficult even if $x \in L$.

4.1.2 Definition of Graph and Hamilton Cycle Problem

A graph $G = (V, E)$ is a pair of a set of vertices V and a set of edges E associated with pairs of vertices, both being assumed finite. Let $V(G) = \{v_1, v_2, \cdots, v_n\}, E(G) = \{e_1, e_2, \cdots, e_m\}, |V| = n, |E| = m$. We will describe an edge between vertex v_i and vertex v_j as $e_{i,j}$. A walk of length k from v_0 to v_k is $P = (V, E)$ with $V = \{v_0, v_1, \cdots, v_k\}, E = \{e_{0,1}, \cdots, e_{k-1,k}\}$. A path is a walk with all different vertices. A cycle is a path which start vertex is equal to the goal vertex. A cycle that visits all vertices of V is called a Hamilton Cycle (HC). A directed graph is a graph where each edge has a direction, in such graph, $e_{i \to j}$ denotes an edge from vertex v_i to vertex v_j. An undirected graph is one where every edge has both $e_{i \to j}$ and $e_{j \to i}$. A simple graph is one which has no self-loop or multiple edges.

The Hamilton Cycle Problem (HCP) is that, given a graph $G = (V, E)$, decide whether G has a HC or not. A graph with a HC is called Hamiltonian Graph. HCP is NP-complete both for directed graphs and undirected graphs.

4.1.3 Generic Colored Matrix Model

Garg et al. defined a generic colored matrix model in [GGH13+] that captures attacks where the adversary is only allowed to add/multiply matrices in the correct order. They represent this restriction as color assigned to left and right of matrix and handle. For every matrix M, the corresponding record is $(h, M, (m, LC), (n, RC))$ where h is handle, M is $m \times n$ matrix, LC is left color, and RC is right color.

Setup. The represent oracle choose an initial set of l colored matrices and assigns to them the handle, inserts into the database the record $\{(h_i, M_i, (m_i, LC_i), (n_i, RC_i))\}_{i=1}^l$, and sends $\{(h_i, (m_i, LC_i), (n_i, RC_i))\}_{i=1}^l$ to the adversary but not the matrix M.

The adversary sends queries through two handles. A represent oracle, who performs generic computation to the adversary, looks up records corresponding the two handles in the database and if their color and row/column size satisfy the order restriction of addition or multiplication then the oracle adds or multiplies the two matrices. If the result record is not in the database then the oracle inserts the record with new handle h' into database. Then the oracle returns a contained or new handle. The adversary being unable to get matrix, he can only adds and multiplies through given handles. The following is the process of representation oracle.

Addition. When the adversary makes query $\mathsf{add}(h_1, h_2)$, the oracle looks up the records corresponding to h_1, h_2 in the database. If such records exists, then if $(m_1, LC_1) = (m_2, LC_2)$ and $(n_1, RC_1) = (n_2, RC_2)$ then the oracle computes their sum $M = M_1 + M_2$. If the database already contains the matrix $(M, (m_1, LC_1), (n_1, RC_1))$ then oracle returns its handle. Otherwise, the oracle assigns a new handle h' and inserts into the database $(h', M, (m_1, LC_1), (n_1, RC_1))$ and returns the new handle.

Multiplication. When the adversary makes query $\mathsf{mult}(h_1, h_2)$, the oracle looks up the records corresponding to h_1, h_2 in the database. If such records exists, then if $(n_1, RC_1) = (m_2, LC_2)$ then the oracle computes their product $M = M_1 M_2$. If the database already contains the matrix $(M, (m_1, LC_1), (n_2, RC_2))$ then oracle returns its handle. Otherwise, the oracle assigns a new handle h' and inserts into the database $(h', M, (m_1, LC_1), (n_2, RC_2))$ and returns the new handle.

4.2 Design Principle

We use a directed hamiltonian graph to construct our witness encryption. First, we consider design of blinding factor which hides plain text. HC passes every vertex in a graph exactly once , so we assign a secret to every vertex and set some value storing the secret to every edge. We generate a blinding factor from these secrets of all vertices and hide a message by the blinding factor and make it a ciphertext. Another components of ciphertext are edge secrets. One who knows a HC can collect all secrets from ciphertext using the knowledge of HC and he can recover the blinding factor. Difficulty of designing such blinding factor is that, during decrypt one who knows HC can generate the blinding factor and during encrypt the encryptor creates the same value without knowledge of HC.

We adopt to use high-dimensional matrix for achieving this restriction of ordering on decryption because of matrix holds non-commutative. We assign an adjacency matrix to each edge, in which we set a secret of starting vertex, and put these adjacency matrices of all edges in the ciphertext. A decryptor who knows the HC can multiply all adjacency matrices in the order of the given HC and can make a product of the secrets of all vertices in the graph. The product value has no order, so the encryptor also can make the same value.

Detail of the design. We name vertices by its index like $1, 2, \cdots, n$ and let $P_{i \to j}$ be a matrix assigned to an edge $e_{i \to j}$. We set a secret s_i of vertex i to the (i, j)-th element in $P_{i \to j}$. An element of (i, j) in a product matrix of two adjacent matrices P_i and P_j is a product $s_i s_j$. By multiplying matrices of edge in order of the given HC which starts and goals at vertex i, the element of (i, i) in the product matrix is $\Pi_{j \in [n]} s_j$. This value is independent of the order of vertices, so encryptor also can make this value from secrets assigning to vertices.

To prevent pulling out a secret s_i in the edge matrix $P_{i \to j}$, it is necessary to randomize edge matrices. For this, we take in a technique of oblivious transfer proposed by Joe Kilian [Kil88]. We assign a randomized matrix R_i and its inverse R_i^{-1} to each vertex and set $R_i^{-1} P_{i \to j} R_j$ be a new edge matrix. When multiply these new matrix of adjacency edge, e.g. $R_i^{-1} P_{i \to j} R_j$ and $R_j^{-1} P_{j \to k} R_k$, the product $R_j R_j^{-1}$ becomes identity matrix, so the connected edge matrices becomes $R_i^{-1} P_{i \to j} P_{j \to k} R_k$. This works sequentially, a product matrix of connected edge matrices in a path that starts vertex i and goals vertex x becomes $R_i^{-1} P_{i \to j} \cdots P_{(x-1) \to x} R_x$.

If $i \neq x$ then, the secret $\Pi_{k=i}^{x-1} s_k$ in the matrix $\Pi_{k=i}^{x-1} P_k$ is randomized, so it is impossible to pull out the secret. The other hand, if $i = x$ then, the product is $R_i^{-1} P_{i\to j} \cdots P_{y\to i} R_i$ and it is possible to pull out a trace $\Pi_{k=i}^{y} s_k$ of this product. This means that this construction has a problem that one who does not know HC can make blinding factor $\Pi_{j\in[n]} s_j$ from traces of partial cycles. We call partial cycle as "short cycle".

Definition 4 (Short Cycle). *We call a cycle "short cycle" if the cycle satisfies:*
1. The cycle does not pass all vertices in a graph.
2. Each vertices in the cycle is passed only once.

In other words, "short cycle" is a cycle except HC.

We assign one more secret r_i to each vertex i and set that to the $(1,1)$-th element of every edge matrix $P_{i\to j}$. A $(1,1)$-th element of a product matrix of edge matrices in a cycle that starts and goals vertex 1 becomes $\Pi r_i + \Pi s_i$. We regard HC as a cycle that starts and goals vertex 1, so a element $(1,1)$ in product of edge matrices in HC becomes $\Pi_{j\in[n]} r_j + \Pi_{j\in[n]} s_j$. This value is unable to be calculated from traces, that are $\Pi r + \Pi s$ from a short cycle not including vertex 1.

For the purpose of security proof, we assign another matrix R_1' instead of R_1^{-1} s.t. $R_1' R_1 \neq$ identity matrix. In a product of in-edge matrix and out-edge matrix about vertex 1, $R_1^{-1} R_1'$ is not canceled, so it prevent pulling out a trace. The blinding factor is now changed to $R_1'(\Pi_{j\in[n]} r_j + \Pi_{j\in[n]} s_j) R_1$.

Every values assigned to vertex (i.e., secrets s_i, r_i and all elements of random matrices R_i, R_i^{-1} and R_1') are encoded at level-0 by sampling algorithm of multilinear maps. During making edge matrix in encrypt algorithm, all components in edge matrix $R_i^{-1} P_{i\to j} R_j$ are encoded to level-1 and rerandomized, the rerandomized encoding keeps additive and multiplicative homomorphism but does not keep homomorphism divisionally. We set up this encoding system with level-n. Using this zero-test parameter, we can extract a deterministic string from level-n encoding of each element in $R_1'(\Pi_{j\in[n]} r_j + \Pi_{j\in[n]} s_j) R_1$. We concatenate these extracted values and extract a random value of this concatenation by a strong randomness extractor. The extracted value is finally our blinding factor.

4.3 Our Construction

In our scheme of witness encryption, a NP-complete language is Hamilton Cycle Problem, and a statement is a graph G. As building blocks, we use a multilinear map MM and a randomness extractor ext. Notations: The symbol $||$ denotes to concatenate strings. We omit "params" for simplicity in the following algorithms.

4.3.1 Concrete Construction

Encryption. Encryption algorithm takes as input security parameter λ, a graph G of n vertices as statement of language, and a message m. First, it generates parameters of MM specifying level-n. Next, creates a level-0 encoded random matrix R_i and samples two level-0 encodings r_i and s_i for every vertices i in G. Then it creates matrix $P_{i\to j}$ for edge $e_{i\to j}$ and set r_i and s_i to the matrix. A public matrix $\hat{E}_{i\to j}$ is product of R_i^{-1}, $P_{i\to j}$ and R_i, whose elements are encoded level-1. Last, it creates blinding factor from all s_i and r_i and hides a message m.

$\mathsf{Encrypt}(1^\lambda, G, m) \to CT$
  $(\mathsf{params}, p_{zt}) \leftarrow \mathsf{MM.InstGen}(1^\lambda, 1^n)$
  for $i \in [n]$ // `all vertices`
    $R_i \leftarrow \mathsf{sampMatrix}(n)$
    if $i = 1$ then $R_1' \leftarrow \mathsf{sampMatrix}(n)$
    else create a invert matrix R_i^{-1} from R_i
    $r_i \leftarrow \mathsf{samp}()$ $s_i \leftarrow \mathsf{samp}()$
  for $i \in [n]$
    for j s.t. edges $i \to j$ are out-edges of vertex i in G
      create a $n \times n$ matrix $P_{i\to j}$
      $(P_{i\to j})_{1,1} := r_i$ $(P_{i\to j})_{i,j} := s_i$
      $(P_{i\to j})_{x,y} := 0$ if $(x,y) \neq (1,1)$ and (i,j)
      if $i = 1$ then $E_{i\to j} := R_1' P_{1\to j} R_j$ // `level-0`
      else $E_{i\to j} := R_i^{-1} P_{i\to j} R_j$ // `level-0`
      $\hat{E}_{i\to j} \leftarrow \mathsf{encodeMatrix}(1, E_{i\to j})$ // `level-1`
  Create an $n \times n$ matrix P'
  $(P')_{1,1} := \Pi_{i\in[n]} r_i + \Pi_{i\in[n]} s_i$
  $(P')_{x,y} := 0$ if $(x,y) \neq (1,1)$
  for $i, j \in [n]$
    $u := u || \mathsf{MM.ext}(p_{zt}, \mathsf{MM.enc}(n, (R_1' P' R_1)_{i,j}))$
  $C := m \oplus \mathsf{ext}(u)$
  return $CT = (C, \{\hat{E}_{i\to j}\}_{(i,j)\in E(G)}, G)$

$\mathsf{sampMatrix}(n) \to M$
  create a $n \times n$ matrix M
  for $i, j \in [n]$
    $M_{i,j} \leftarrow \mathsf{MM.samp}()$ // `level-0`
  return M

$\mathsf{encodeMatrix}(k, M) \to M$
  for $i, j \in [n]$
    $M_{i,j} \leftarrow \mathsf{MM.reRand}(k, \mathsf{MM.enc}(k, M_{i,j}))$ //`level-k`
  return M

Decryption. Decryption algorithm takes as input a ciphertext CT and a hamilton cycle HC as witness. We can suppose the first vertex of HC is vertex 1. The algorithm chooses edge matrix in CT in the order of HC and multiplies them. Then it extracts a deterministic string for every element in the product by $\mathsf{MM.ext}$ and it concatenates all of the strings. Here $j = \mathsf{HC}(i)$ denotes a next vertex to i in HC $\{\cdots, i, j, \cdots\}$.

$\mathsf{Decrypt}(CT, HC)$
  $M := \hat{E}_{1\to HC(1)} \hat{E}_{HC(1)\to HC^2(1)} \cdots \hat{E}_{HC^{n-1}(1)\to 1}$
  for $i, j \in [n]$
    $u := u || \mathsf{MM.ext}(p_{zt}, \mathsf{MM.enc}(n, M_{i,j}))$
  return $m := C \oplus \mathsf{ext}(u)$

4.3.2 Correctness

Now we show a blinding factor u recovered in the decryption algorithm is the same as u in the encryption algorithm. First, we evaluate M in the decryption. Every element in M is rerandomized encoding and we can homomorphically add and multiply them. So M is equivalent to level-n encoding of the following product.

$$(R_1' P_{1\to 2} R_2)(R_2^{-1} P_{2\to 3} R_3) \cdots (R_{(n-1)}^{-1} P_{(n-1)\to 1} R_1) = R_1' P_{1\to 2} P_{2\to 3} \cdots P_{(n-1)\to 1} R_1$$

Let P'' be $P_{1\to 2} P_{2\to 3} \cdots P_{(n-1)\to 1}$, then $P''_{1,1} = \Pi_{i\in[n]} r_i + \Pi_{i\in[n]} s_i$ and $P''_{x,y} = 0$ for $(x,y) \neq (1,1)$. So P'' is equal to P' calculated in the encryption algorithm. For each element in P' and P'', their output values by $\mathsf{MM.ext}$ are the same.

Therefore the blinding factor made by the encryption algorithm and the decryption algorithm are the same.

4.4 Proof of Security: Soundness Security

First we define a variant of generic colored matrix model: *a generic cyclic colored matrix model*, and then define a security game based on the model.

4.4.1 Generic Cyclic Colored Matrix Model

We define a generic cyclic colored matrix model by expanding a generic colored matrix model defined by Garg et al. in [GGH13+] that captures attacks where the adversary only adds and multiplies matrices in the correct order.

A matrix in our ciphertext is sandwiched in between two random matrices, when multiply adjacent edge matrices, such random matrices are canceled. But when multiply not-adjacent edge matrices, they are not canceled and randomization remains . The generic colored matrix model [GGH13+] modeled this situation as colored matrix: an adversary multiply matrices only if he queries two matrices such that RC_1 is the same as LC_2. Since our ciphertexts are encoded, we can add and multiply elements in matrices homomorphically but we can not divide one by another. In the generic colored matrix model [GGH13+], a adversary queries add and mult using handle, so, this models our encoding restriction.

If the LC and RC of a result matrix M are same then a trace of matrix M is possible to be pulled out. In the oracle's process in our model, when such case, the oracle computes a trace of the result matrix M, and returns a record additionally for the trace. In our model, there are two record types: "Matrix Type" and "Scalar Type". We add a query $\mathsf{scalar\_mult}$ that multiplies a trace value in a "Scalar Type" record by a matrix in a "Matrix Type" record.

We does not return a handle of a eigenvalue of a matrix, because computing the eigenvalue needs division during Gaussian elimination, but the division is not provided on the encoding system used in our scheme.

Detail of The Generic Cyclic Colored Matrix Model Setup and Addition algorithms are the same as the original generic colored matrix model. In Multiplication algorithm, the following process is executed after Multiplication process of the original model. If the types of record corresponding h_1 and h_2 are both matrix type and LC_1 and RC_2 of result matrix are the same then the oracle computes a trace M' of the result matrix. If the trace is in the database then returns its handle, otherwise inserts into the database the new record with new handle $(h'', M', (1,0), (1,0))$ and returns its handle.

Scalar Multiplication. When the adversary makes query $\mathsf{scalar\_mult}(h_1, h_2)$, the oracle looks up the records corresponding to h_1, h_2 in the database. If such records exists, we view the first record as matrix type and the second record as scalar type, then the oracle computes product $M = \text{trace} \times M_1$. If the database already contains the matrix $(M, (m, LC), (n, RC))$ then oracle returns its handle. Otherwise, the oracle assigns a new handle h' and inserts into the database $(h', M, (m_1, LC_1), (n_1, RC_1))$ and returns the new handle.

4.4.2 Definition of Security Game for Witness Encryption using Multilinear Maps

We define a security game based on the generic cyclic colored matrix model for our witness encryption using multilinear maps. In our witness encryption, a matrix assigned to each edge corresponds to one record in the generic cyclic colored matrix model. To multiply two colored matrices$(h_1, M_1, (m_1, LC_1), (n_1, RC_1))$ and $(h_2, M_2, (m_2, LC_2), (n_2, RC_2))$ by $\mathsf{mult}(h_1, h_2)$ means to make a path by connecting two edges, and if $LC_1 = RC_2$, it means that the start vertex and the goal vertex are the same, in other words, a path becomes a cycle in our scheme. So, in the security game, the representation oracle returns a handle of trace if the result of $\mathsf{mult}(h_1, h_2)$ becomes a cycle. Exceptionally, a random matrix R_1', which is not a inverse of R_1, is assigned to out-edges of vertex 1, so the left and right colors are different and the oracle does not return a handle of trace in this case.

Details of the Security Game for our Witness Encryption.

Setup. The oracle is given a graph G, it makes matrix type records for all edges in G and inserts into database them and returns their representations $\{(h, (m, LC), (n, RC))\}$ without giving matrix M itself to the adversary. To say more details, the oracle samples two level-0 encoded secrets r_i and s_i by $\mathsf{MM.samp}()$ for each vertex i. For every edge $e_{i\to j}$, it sets the secrets to a adjacent matrix $M_{i\to j}$ where $(M_{i\to j})_{1,1} = r_i$ and $(M_{i\to j})_{i,j} = s_i$, and inserts into the database those records $\{(h, M_{i\to j}, (n, i), (n, j))\}$. Then it gives $\{(h, (n, i), (n, j))\}$ to the adversary.

Addition. When the adversary makes query $\mathsf{add}(h_1, h_2)$, the oracle looks up the records corresponding to h_1, h_2 in the database. If such records exists then, if $(m_1 \neq m_2)$ or $(LC_1 \neq LC_2)$ or $(n_1 \neq n_2)$ or $(RC_1 \neq RC_2)$ then return bot. Otherwise, it adds $M = M_1 + M_2$ and looks up the record $(M, (m, LC), (n, RC))$, if it exists in the database then returns the handle, otherwise the oracle inserts the record $(h, M, (m, LC), (n, RC))$ with new handle h and returns the handle.

Multiplication. When the adversary makes query $\mathsf{mult}(h_1, h_2)$, the oracle looks up the records corresponding to h_1, h_2 in the database. If such records exists, if $n_1 \neq m_2$ or $RC_1 \neq LC_2$ then return bot. Otherwise, it multiplies $M = M_1 M_2$ and looks up the record $(M, (m_1, LC_1), (n_2, RC_2))$, if it exists in the database then returns the handle, otherwise the oracle inserts the record $(h, M, (m_1, LC_1), (n_2, RC_2))$ with new handle h and returns the handle. If the matrices corresponding to h_1, h_2 are both matrix type and they starts and goals at same vertex such that the vertex is not 1 then the oracle computes a trace of M and puts the trace to 1×1 matrix M and inserts into the scalar type record $(h, M, (1,0), (1,0))$ and returns its handle to the adversary.

Scalar Multiplication. When the adversary makes query $\mathsf{scalar\_mult}(h_1, h_2)$, the oracle looks up the records corresponding to h_1, h_2 in the database. If such records exists, let first record to be matrix type, second record to be scalar type, then the oracle multiplies $M = (M_2)_{1,1} M_1$ and looks up the record $(M, (m_1, LC_1), (n_1, RC_1))$, if it exists in the database then returns the handle, otherwise the oracle inserts the record $(h, M, (m_1, LC_1), (n_1, RC_1))$ with new handle h and returns the handle.

4.4.3 Proof of Security

THEOREM 2. *Our witness encryption based on HC is soundness secure in the generic cyclic colored matrix model.*

More precisely, the probability that an adversary distinguishes a real model and a simulation is bounded above T^3/p from the Schwartz-Zippel lemma [S80] [Z79] where an adversary receives at most T handles from the oracle and the secrets r, s are independently and uniformly chosen from $\mathbb{Z}_p$. T^3 means a number of handles $\times$ a number of combinations of r, s.

Proof. We make simulator which input is a graph which does not have any HC, and show that there is no handle of the matrix corresponding to the "full cycle":$\Pi_{j\in[n]}r_j + \Pi_{j\in[n]}s_j$ in a view of the adversary.

Algorithm of Simulator. We assume there are t short cycles which does include vertex 1 in a given graph G. Let sc be a set of vertices of a short cycle. Input values to a simulator is a graph G. When simulator needs to return a handle of trace, it calculates a trace from the result matrix, then returns the result's handle.
Notations:
A given graph G has n vertices. A record rec_i denotes the record $(h_i, M_i, (m_i, LC_i), (n_i, RC_i))$ in the database.

Setup. A simulator makes edge matrices for all edges in given graph G, and inserts into the database those edge matrices and sends representations of edge without matrix M to the adversary.

for $i \in [n]$ // all vertices
create a variable R_i S_i
for j s.t. edges $i \to j$ are out-edges of vertex i
create a $n \times n$ matrix $M_{i\to j}$
$(M_{i\to j})_{1,1} := R_i$ $(M_{i\to j})_{i,j} := S_i$
$(M_{i\to j})_{x,y} := 0$ if $(x, y) \neq (1, 1)$ and (i, j)
$h_k :=$ new handle
$\mathsf{db.insert}(h_k, M_{i\to j}, (n, i), (n, j))$
give $\{(h_k, (n, i), (n, j))\}$ to the adversary.

Addition. If types of associated h_1, h_2 are both matrix type, if their start vertices are same and their goal vertices are also same then the simulator adds two matrices and inserts its record into database and returns the handle.

$\mathsf{add}(h_1, h_2)$
$rec1 = \mathsf{db.select}(h_1)$ $rec2 = \mathsf{db.select}(h_2)$
if $(m_1 \neq m_2)$ or $(LC_1 \neq LC_2)$ or $(n_1 \neq n_2)$ or $(RC_1 \neq RC_2)$ then return $\perp$
$M = M_1 + M_2$
$h \leftarrow \mathsf{db.select}(M, (m_1, LC_1), (n_1, RC_1))$
if h does not exist in database then
$h :=$ new handle
$\mathsf{db.insert}(h, M, (m_1, LC_1), (n_1, RC_1))$
return h

Multiplication. If types of associated h_1, h_2 are both matrix type, if first goal vertex and second start vertices are same, and first col size and second row size are same then the simulator multiplies the two matrices. If start and goal are same but not 1 then it calculates a trace from the result matrix M, and inserts its record into database and returns the handle.

$\mathsf{mult}(h_1, h_2)$
$rec1 = \mathsf{db.select}(h_1)$ $rec2 = \mathsf{db.select}(h_2)$
if $(n_1 \neq m_2)$ or $(RC_1 \neq LC_2)$ then return $\perp$
$M = M_1 \cdot M_2$
$h \leftarrow \mathsf{db.select}(M, (m_1, LC_1), (n_2, RC_2))$
if h does not exist in database then
$h :=$ new handle
$\mathsf{db.insert}(h, M, (m_1, LC_1), (n_2, RC_2))$
if types of h_1, h_2 are matrix type and $(LC_1 = RC_2)$ and $(LC_1 \neq 1)$ then
create 1×1 matrix M_t
$(M_t)_{1,1} = M_{1,1} + M_{LC_1,RC_2}$
$h_t \leftarrow \mathsf{db.select}(M_t, (1, 0), (1, 0))$
if h_t does not exist in database then
$h_t :=$ new handle
$\mathsf{db.insert}(h_t, M_t, (1, 0), (1, 0))$
return (h, h_t)
else return h

Scalar Multiplication. It multiplies a trace value in scalar type record by a matrix in a matrix type record and inserts the result record into database and returns the handle.

$\mathsf{scalar\_mult}(h_1, h_2)$
$rec1 = \mathsf{db.select}(h_1)$ $rec2 = \mathsf{db.select}(h_2)$
let $rec1$ be a $n \times n$ edge matrix.
let $rec2$ be a 1×1 trace matrix.
if not above then return $\perp$
$M = (M_2)_{1,1} \cdot M_1$
$h \leftarrow \mathsf{db.select}(M, (m_1, LC_1), (n_1, RC_1))$
if h does not exist in database then
$h :=$ new handle
$\mathsf{db.insert}(h, M, (m_1, LC_1), (n_1, RC_1))$
return h

Analysis. We discuss about difference of distributions of adversary's view between real model and simulation. The simulator replaces random elements of edge matrices into variables, so it's possible that result of calculation of the elements is in database at real model but not in database at simulation. In such a case, he returns existing handle at real model and returns new handle at simulation, so the view of adversary is different. But its provability is bounded above T^3/p from the Schwartz-Zippel lemma [S80] [Z79] where an adversary receives at most T handles from the oracle and the secrets r, s are independently and uniformly chosen from any finite set$S, |S| = p$.

We show there is no record of our blinding factor in a simulator's database. Now we recall that our blinding factor is concatenation of all elements in the product matrix $R'_1(\Pi_{j\in[n]}r_j + \Pi_{j\in[n]}s_j)R_1$. We confirm about edge matrix records at each query. After setup, value in matrix column is M s.t. $M_{1,1} = R_i$ and $M_{i,j} = S_i$ for every edge $e_{i\to j}$. When two matrices M_1 and M_2 are added by add query, the result matrix M is such that $M_{1,1} = (M_1)_{1,1} + (M_2)_{1,1}, M_{i,j} = (M_1)_{i,j} + (M_2)_{i,j}$ and other elements are all zero. When two matrices M_1 and M_2 are multiplied by mult query, if $LC_1 \neq 1'$ or $RC_2 \neq 1$ then the result matrix M is such that $M_{1,1} = (M_1)_{1,1}(M_2)_{1,1}, M_{i,k} = (M_1)_{i,j}(M_2)_{j,k}$ and other elements are all zero. If $LC_1 = 1'$ and $RC_2 = 1$ then the result matrix M is such that $M_{1,1} = (M_1)_{1,1}(M_2)_{1,1} + (M_1)_{i,j}(M_2)_{j,k}$ and other elements are all zero. When a trace v and matrix M_1 are multiplied by $\mathsf{scalar\_mult}$, the result matrix M is such that $M_{1,1} = v(M_1)_{1,1}, M_{i,j} = v(M_1)_{i,j}$ and other elements are all zero. We notice that elements in M_1 and

M_2 in above queries may be multiplied by some traces by scalar_mult queries.

Now we consider whether there is a record of matrix M' s.t. $M'_{1,1} = \Pi_{i\in[n]}R_i + \Pi_{i\in[n]}S_i$ and other elements are all zero for our blinding factor.

CLAIM 4. *In the database, if there is records where $LC = 1'$ and $RC = 1$ and $M_{1,1}$ has some value v and $M_{i,j}$ $(i, j \neq 1)$ is zero then*

$$v = \Sigma_{sc}(\Pi_{sc'_i}(\Pi_{j\in sc'_i}R_j + \Pi_{j\in sc'_i}S_j))(\Pi_{j\in sc}R_j + \Pi_{j\in sc}S_j), \cdots Eq(1)$$

where sc denotes a short cycle which starts and goals at vertex 1, sc' denotes a short cycle which does not include vertex 1.

Proof. A trace is registered into database when a short cycle which does not include vertex 1 is made by mult query. After the trace is registered, a matrix may be scalar multiplied by scalar_mult query. Those matrices may be added and multiplied by add and mult. Using such two matrices, when a short cycle which starts and goals at vertex 1 is made by mult, a record where $LC = 1'$ and $RC = 1$ and $M_{1,1}$ has some value v and $M_{i,j}$ $(i, j \neq 1)$ is zero is registered. The form of v is $v = (\Sigma_{sc}\Pi_{sc'_i}(\Pi_{j\in sc'_i}R_j + \Pi_{j\in sc'_i}S_j))(\Pi_{j\in sc}R_j + \Pi_{j\in sc}S_j)$. From the observation before Claim 4, add and scalar_mult does not change the color and element position of the result matrix, so records where $LC = 1'$ and $RC = 1$ must be made from records where $LC = 1'$ and $RC = 1$ by add or scalar_mult. The result $M_{1,1}$ has a form $v = \Sigma_{sc}(\Pi_{sc'_i}(\Pi_{j\in sc'_i}R_j + \Pi_{j\in sc'_i}S_j))(\Pi_{j\in sc}R_j + \Pi_{j\in sc}S_j)$. □

CLAIM 5. *The Eq(1) never equals to $\Pi_{i\in[n]}R_i + \Pi_{i\in[n]}S_i$.*

Proof. To make $\Pi_{i\in[n]}R_i + \Pi_{i\in[n]}S_i$, it is necessary to multiply one $(\Pi_{j\in sc}R_j + \Pi_{j\in sc}S_j)$ and some $(\Pi_{j\in sc'_i}R_j + \Pi_{j\in sc'_i}S_j)$ because the input graph does not contain any Hamilton Cycle. The result has cross terms of R_jS_j, and for canceling this cross terms it needs a new product of $(\Pi_{j\in sc}R_j + \Pi_{j\in sc}S_j)$ and $(\Pi_{j\in sc'_i}R_j + \Pi_{j\in sc'_i}S_j)$ of smaller short cycles. Then the result has their new cross terms. To cancel those cross terms recursively, finally it becomes sums of products of $(\Pi_{j\in sc}R_j + \Pi_{j\in sc}S_j)$ and $(\Pi_{j\in sc'_i}R_j + \Pi_{j\in sc'_i}S_j)$ of minimum short cycles, and their cross terms are unable to be canceled. Therefor the Eq(1) never equals to $\Pi_{i\in[n]}R_i + \Pi_{i\in[n]}S_i$. □

Now the proof of Theorem 2 is complete. □

5. CONCLUSION

We proposed two applications of multilinear maps, GKE and witness encryption. The communication structure of these schemes are both formed like that some pre-specified number of points are connected by communication lines, and these schemes share a common mechanism. The mechanism is that, each point has a secret level-0 encoding and its level-1 encodings are piled up in a product one by one along the lines. Eventually a specified level encoding is generated, and a deterministic value is extracted from the encoding.

The security of our GKE scheme is based on GDDH assumption. Each party generates a level-(N-1) encoding, in its source encodings only its own encoding is level-0 and others are level-1, and extracts a same group key from that. The piling encodings mechanism is useful in some cases to reduce a lot of traffic in the one-round GKE or existing GKE that does not use multilinear maps. In our witness encryption scheme, we expressed Hamilton Cycle, that visits each vertex in the graph exactly once, using this piling encodings mechanism. We proved its soundness security based on our Generic Cyclic Colored Matrix Model.

6. REFERENCES

[BCPQ] E. Bresson, O. Chevassut, D. Pointcheval, and J. J. Quisquater. Provably authenticated group diffie-hellman key exchange. In proceedings of 8th ACM Conference on CCS E., pages 255-264, 2001.

[BS03] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. In Contemporary Mathematics 324, pages 71-90, 2003.

[CLT13] Jean-S é bastien Coron, Tancr è de Lepoint, Mehdi Tibouchi. Practical Multilinear Maps over the Integers. In CRYPTO 2013, pages 476-493.

[GGH13] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices and applications. In EUROCRYPT 2013, Lecture Notes in Computer Science. Springer, 2013. Cryptology ePrint Archive, Report 2012/610.

[GGH13+] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, Brent Waters. Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits. In FOCS 2013, pages 40-49.

[GGHSW13] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attributebased encryption for circuits from multilinear maps. In Cryptology ePrint Archive, Report 2013/128, 2013.

[GGSW] Sanjam Garg, Craig Gentry, Amit Sahai and Brent Waters. Witness Encryption and its Applications. In STOC, pages 467-476, 2013.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In Janos Simon, editor, In STOC, pages 20-31. ACM, 1988.

[S80] T.Schwartz. Fast probabilistic algorithms for verification of polynomial identities. In Journal of the ACM 27: pages 701 – 717, 1980.

[Shoup] V. Shoup. Sequences of Games: A Tool for Taming Complexity in Security Proofs. In Cryptology ePrint Archive, Report 2004/332, 2004.

[STW96] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman Key Distribution Extended to Group Communication. In Proceedings of the 3rd ACM Conference on Computer and Communications Security, pages 31-37. ACM Press, 1996.

[Z79] R. Zippel. Probabilistic algorithms for sparse polyno-mials. In Proceedings of EUROSAM, Springer Lecture Notes in Computer Science Vol.72, pages 216 – 226, 1979.