UNIVERSALLY COMPOSABLE NON-COMMITTING ENCRYPTIONS IN THE PRESENCE OF ADAPTIVE ADVERSARIES

Huafei Zhu[1], Tadashi Araragi[2], Takashi Nishide[3] and Kouichi Sakurai[3]
[1]*Institute for Infocomm Research, A-STAR, Singapore*
[2]*NTT Communication Science Laboratories, Kyoto, Japan*
[3]*Department of Computer Science and Communication Engineering, Kyushu University, Fukuoka, Japan*

Keywords: Adaptive security, Decisional Diffie-Hellman assumption, Non-committing encryptions, Oblivious sampling and faking algorithms.

Abstract: Designing non-committing encryptions tolerating adaptive adversaries is a challenging task. In this paper, a simple implementation of non-committing encryptions is presented and analyzed in the strongest security model. We show that the proposed non-committing encryption scheme is provably secure against adaptive adversaries in the universally composable framework assuming that the decisional Diffie-Hellman problem is hard.

1 NON-COMMITTING ENCRYPTIONS

Informally, a non-committing encryption protocol is an encrypted communication that allows a simulator to open a ciphertext to any plaintext it desires and simulate the real world adversary's view before and after a player is corrupted. Nielsen (Nielsen, 2002) shows that no non-interactive communication protocol can be adaptively secure in the asynchronous model.

Beaver and Haber's protocol (Beaver and Haber, 1992) realizes the functionality of non-commitment encryption schemes in the erasure model. That is, if one is willing to trust honest parties can erase sensitive information such that the adversary can find no trace of it, should he break in, then such adaptively secure multi-party computation of any function can be efficiently realized. Subsequently, Beaver (Beaver, 1997) proposed a much simpler scheme based on the decisional Diffie-Hellman assumption with expansion factor $O(k)$. The non-committing encryptions presented in (Beaver, 1997) and (Beaver and Haber, 1992) are designed and analyzed in the stand-alone, simulation-based framework.

Canetti, Feige, Goldreich and Naor (Canetti et al., 1996) proposed the first non-committing encryptions based on so called common-domain permutations (the stand-alone non-committing encryption presented in (Canetti et al., 1996) is secure against adaptive adversary in the universally composable framework, see Section 6.3 of (Canetti, 2005) for more details). To encrypt 1 bit, $\Theta(k^2)$ public key bits are communicated. Damgård and Nielsen (Damgård and Nielsen, 2000) proposed generic constructions of non-committing encryption schemes based on so called simulatable public-key encryption schemes in the universally composable framework (a detailed analysis of the protocol presented in (Damgård and Nielsen, 2000) is available in Section 4 of (Nielsen, 2003)). Roughly speaking, a public-key encryption scheme is simulatable if, in addition to the normal key generation algorithm procedure, there is an algorithm to generate a public key without knowing the corresponding secret key. Moreover, it must be possible to sample efficiently a random ciphertext without getting to know the corresponding plaintext. They showed that a non-committing encryption scheme can be constructed from any semantically secure and simulatable public-key system. Although the Damgård and Nielsen's construction (Damgård and Nielsen, 2000) is general, the cost of computation is expensive since one should obliviously generate a pair of public keys to communicate 1-bit in open networks.

Very recently, Choi, Soled, Malkin and Wee (S.Choi et al., 2009) have presented a new implementation of non-committing encryptions based on a weaker notion (called trapdoor simulatable cryptosystems). The idea behind their construction is that − on input a security parameter k, a receiver first generates total $4k$ public keys where the first k public keys are generated by a key generation algorithm of the underlying trapdoor simulatable encryption scheme while the rest $3k$ public keys are generated by an oblivious sampling algorithm. To encrypt a bit b, the sender sends $4k$ ciphertexts of which k are encrypted b and the remaining $3k$ ones are obliviously sampled. Although the non-committing encryption scheme in (S.Choi et al., 2009) is at the expense of higher computation and communication of the Damgård and Nielsen's protocol (Damgård and Nielsen, 2000), such an implementation is definitely interesting since the subtle failure model in (Damgård and Nielsen, 2000) is eliminated (i.e., the scheme presented in (S.Choi et al., 2009) is round-optimal) in their framework.

1.1 This Work

This paper studies non-committing encryptions in the UC-framework of Canetti. We will show that the proposed non-committing encryption scheme is provably secure against adaptive adversaries in the universally composable framework assuming that the decisional Diffie-Hellman problem is hard.

An Overview of the Protocol. The proposed non-committing encryption protocol comprises two phases: a channel setup phase and a communication phase. The idea behind our construction is simple: to set up a secure channel, a sender S first picks a random bit $\alpha \in \{0,1\}$, and then selects a Diffie-Hellman quadruple e_α and a garbled quadruple $e_{1-\alpha}$ and send (e_0, e_1) to a receiver R. Given (e_0, e_1), the receiver R picks a selection string f_β and a garbled string $f_{1-\beta}$, and then obliviously selects 1-out-of-2 quadruples with the help of the selection string f_β. If e_α is selected, then a secure channel is established; Otherwise, S and R retry the channel setup procedure.

Main Result. We claim that the non-commitment protocol π presented in Section 4 realizes the universally composable security in the presence of adaptive adversaries assuming that the Decisional Diffie-Hellman problem is hard.

The Proof of Security. We will show that for any real world adversary $\mathcal{A}$ there exists an ideal-world adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and players running π, or with $\mathcal{S}$ and $\mathcal{F}_{\mathrm{SC}}^{\mathcal{N}}$ in the ideal execution if the decisional Diffie-Hellman assumption holds. The core technique applied to the security proof is a novel application of oblivious sampling and faking algorithms introduced and formalized by Canetti and Fischlin in (Canetti and Fischlin, 2001). Roughly speaking, an oblivious faking algorithm fake takes $g \in G$ as input and outputs a string $r_g \in \{0,1\}^{2|p|}$. An oblivious sampling algorithm sample takes $r \in_U \{0,1\}^{2|p|}$ as input and outputs an element $r_G \in G$. The oblivious sampling and faking algorithms engaged in the security proof benefit a PPT simulator to generate subgroup elements of $G \subseteq Z_p^*$ uniformly at random and interprets a Diffie-Hellman quadruple e_α as a garbled quadruple $e_{1-\alpha}$. The oblivious sampling and faking algorithms also benefit the simulator to interpret a random selection string as a garbled string. As a result, no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and players running π, or with $\mathcal{S}$ and $\mathcal{F}_{\mathrm{SC}}^{\mathcal{N}}$ in the ideal execution if the decisional Diffie-Hellman assumption holds.

Efficiency. Our scheme requires 3 messages to communicate k encrypted bits, where k is the security parameter. The total communication is $O(k)$ Diffie-Hellman quadruples and garbled quadruples and $O(k)$ selection strings and garbled strings and k bits (the communication of the final k bits of the communication depend on the actual messages to be sent). Thus, our universally composably secure non-committing encryption protocol is as efficient as the stand-alone, simulation-based (but the notion of environment is defined in their security definition and the proof of the protocols) protocol by Beaver (Beaver, 1997) − the most efficient implementation of non-committing encryptions so far.

Road-map. The rest of this paper is organized as follows: In Section 2, the building blocks are sketched;The functionality and security definition of non-committing encryption protocols are presented in Section 3. In Section 4, a new non-committing encryption scheme is proposed and analyzed in the universally composable framework in the presence of adaptive adversaries. We conclude our work in Section 5.

2 PRELIMINARIES

We assume that a reader is familiar with the standard notion of universally composable framework (Canetti, 2001). The oblivious sampling and faking algorithms described below are due to Canetti and Fischlin (Canetti and Fischlin, 2001). The two algorithms combined together allow a simulator to construct a fake transcript to the environment $\mathcal{Z}$ in such a way that the simulator can open this transcript to the actual inputs that the simulator receives from the functionality when the parties get corrupted, a core task to prove the security of protocols against adaptive adversaries in the universally composable security model.

The Canetti-Fischlin Oblivious Sampling Algorithm. Let $p = wq + 1$ for some w not divisible by q, and G is a cyclic group of order q in Z_p^*. The Canetti-Fischlin oblivious sampling algorithm sample takes $r \in \{0,1\}^{2|p|}$ as input and outputs an element $r_G \in G$ via the following computations

- the sampling algorithm sample chooses a string $r \in \{0,1\}^{2|p|}$ uniformly at random, where $|p|$ be the bit length of the prime number p.
- Let $r_p = r \bmod p$ and $r_G = r_p^w \bmod p$.

Lemma 1 (Due to (Canetti and Fischlin, 2001)). Let $X = [X = x : x \in_U G]$, and $Y = [Y = y : y \leftarrow \mathsf{sample}(r), r \in_U \{0,1\}^{2|p|}]$, then the distributions between two random variables X and Y are statistically indistinguishable.

The Canetti-Fischlin Oblivious Faking Algorithm. Let $p = wq + 1$ for some w not divisible by q, and G is a cyclic group of order q in Z_p^*. The Canetti-Fischlin oblivious faking algorithm fake takes a random element $h \in G$ as input and outputs $r_h \in \{0,1\}^{2|p|}$ via the following computations

- On input $h \in G$, the faking algorithm fake picks a random integer $i \in Z_w$. Let $h_p = h^x g^{iq} \bmod p$, where $xw \equiv 1 \bmod q$;
- Let $r_h = \mathrm{Len}(\sum_{j \in Z_p} jp + h_p)$, where $\mathrm{Len}(x)$ denotes the bit length of an integer x.

Lemma 2 (due to (Canetti and Fischlin, 2001)) Let $X = [X = x : x \in_U \{0,1\}^{2|p|}]$, and $Y = [Y = y : y \leftarrow \mathsf{fake}(g), g \in_U G]$, then the distributions between two random variables X and Y are statistically indistinguishable.

2.1 The Decisional Diffie-Hellman Assumption

Let $p = 2q + 1$ and p, q be large prime numbers. Let $G \subseteq Z_p^*$ be a cyclic group of order q. Let g be a random generator of G. For any $0 \neq x \in Z_q$, we define $\mathsf{DLog}_G(x) = \{(g, g^x) : g \in G\}$. On input $(g_1, h_1) \in \mathsf{DLog}_G(x_1)$, and $(g_2, h_2) \in \mathsf{DLog}_G(x_2)$, a mapping ϕ called Naor-Pinkas randomizer is defined below:

$$\phi((g_1, g_2, h_1, h_2) \times (s,t)) = (g_1^s g_2^t \bmod p, h_1^s h_2^t \bmod p)$$

where $s, t \in_U Z_q$

Denote $u = g_1^s g_2^t \bmod p$ and $v = h_1^s h_2^t \bmod p$. Naor and Pinkas (Moni Naor, 2001) have shown that

- if $x_1 = x_2$ ($=x$), then (u,v) is uniformly random in $\mathsf{DLog}_G(x)$;
- if $x_1 \neq x_2$, then (u,v) is uniformly random in G^2.

3 NON-COMMITTING ENCRYPTIONS: FUNCTIONALITY AND SECURITY DEFINITION

The notion of non-committing encryption scheme introduced in (Canetti et al., 1996) is a protocol used to realize secure channel in the presence of an adaptive adversary. In particular, this means that a simulator can build a fake transcript to the environment $\mathcal{Z}$, in such a way that the simulator can open this transcript to the actual inputs, that the simulator receives from the functionality when the parties get corrupted.

Let $\mathcal{N}$ be a non-information oracle which is a PPT Turing machine that captures the information leaked to the adversary in the ideal-world. That is, $\mathcal{N}$ is the oracle which takes $(\mathrm{Send}, \mathrm{sid}, P, m)$ as input and outputs $(\mathrm{Send}, \mathrm{sid}, P, |m|)$. Let ChSetup be a channel setup command which on inputs $(\mathrm{ChSetup}, \mathrm{sid}, S)$ produces no output and $(\mathrm{Corrupt}, \mathrm{sid}, P)$ be a corruption command which takes $(\mathrm{Corrupt}, \mathrm{sid}, P)$ produces no output. The functionality of non-committing encryption secure channels defined below is due to Garay, Wichs and Zhou (Garay et al., 2009).

The ideal functionality $\mathcal{F}_{\mathrm{SC}}^{\mathcal{N}}$

Channel setup: upon receiving an input $(\mathsf{ChSetup}, \mathrm{sid}, S)$ from party S, initialize the machine $\mathcal{N}$ and record the tuple $(sid, \mathcal{N})$. Pass the message $(\mathsf{ChSetup}, S)$ to R. In addition, pass this message to $\mathcal{N}$ and forward its output to $\mathcal{S}$;

Message transfer: Upon receiving an input $(\mathsf{Send}, \mathrm{sid}, P, m)$ from party P, where $P \in \{S, R\}$, find a tuple $(\mathrm{sid}, \mathcal{N})$, and if none exists, ignore the message. Otherwise, send the message $(\mathsf{Send}, \mathrm{sid}, P, m)$ to the other party $\overline{P} = \{S, R\} \setminus \{P\}$. In addition, invoke N with $(\mathsf{Send}, \mathrm{sid}, P, m)$ and forwards its output $(\mathsf{Send}, \mathrm{sid}, P, |m|)$ to the adversary $\mathcal{S}$.

Corruption: Upon receiving a message $(\mathsf{Corrupt}, \mathrm{sid}, P)$ from the adversary $\mathcal{S}$, send

$(\text{Corrupt}, \text{sid}, P)$ to $\mathcal{N}$ and forward its output to the adversary. After the first corruption, stop execution of $\mathcal{N}$ and give the adversary $\mathcal{S}$ complete control over the functionality.

Definition (due to (Garay et al., 2009)). We call the functionality $\mathcal{F}_{\text{SC}}^{\mathcal{N}}$ a non-committing encryption secure channel. A real-world protocol π which realizes $\mathcal{F}_{\text{SC}}^{\mathcal{N}}$ is called a non-committing encryption scheme.

4 NON-COMMITTING ENCRYPTION

In this section, we first describe an implementation of non-committing schemes based on the decisional Diffie-Hellman problem, and then show that the proposed scheme realizes UC-security in the presence of adaptive adversaries.

4.1 Description of Non-committing Encryption Scheme

The non-committing encryption protocol comprises two phases: a channel setup phase and a communication phase. To set up a secure channel, S prepares two quadruples e_0 and e_1, where e_α is a Diffie-Hellman quadruple and $e_{1-\alpha}$ is a garbled quadruple, $\alpha \in \{0,1\}$. S now let a receiver R to choose 1-out-of-2 quadruples. If e_α is selected, a secure channel has been set up between two parties; Otherwise, S and R retry the channel setup procedure. The details of protocol are depicted below.

Initialization the environment $\mathcal{Z}$ invokes a system key generation algorithm $\mathcal{G}$ which takes security parameter k as input and outputs (p,q,G), where p is a large safe prime number (i.e., p=$2q+1$, q is a prime number) and G is a cyclic group with order q.

Channel setup: To set up a secure channel, a sender S and a receiver R jointly perform the following computations

- On input (p,q,G), S chooses $\alpha \in_U \{0,1\}$ and performs the following computations
  - S generates a random Diffie-Hellman quadruple ($g_{1,\alpha}$, $g_{2,\alpha}$, $h_{1,\alpha}$, $h_{2,\alpha}$), where $g_{1,\alpha}$ and $g_{2,\alpha}$ are two random generators of G, and $h_{1,\alpha}$ and $h_{2,\alpha}$ are two elements in G such that $h_{1,\alpha} = g_{1,\alpha}^{sk_\alpha}$ mod p, and $h_{2,\alpha} = g_{2,\alpha}^{sk_\alpha}$ mod p, where $sk_\alpha \in_U Z_q$. Let e_α = ($g_{1,\alpha}$, $g_{2,\alpha}$, $h_{1,\alpha}$, $h_{2,\alpha}$).
  - S picks a quadruple ($g_{1,1-\alpha}$, $g_{2,1-\alpha}$, $h_{1,1-\alpha}$, $h_{2,1-\alpha}$) $\in G^4$ uniformly at random. Let $e_{1-\alpha}$ =($g_{1,1-\alpha}$, $g_{2,1-\alpha}$, $h_{1,1-\alpha}$, $h_{2,1-\alpha}$).
  - S sends (e_0, e_1) to R.
- Upon receiving (e_0, e_1), R checks the conditions $1 \neq g_{i,j} \in G$ and $1 \neq h_{i,j} \in G$ (i =1, 2, j =0, 1), if any of the conditions are violated, R outputs $\perp$; Otherwise, R performs the following computations
  - R chooses $\beta \in_U \{0,1\}$ and $s_\beta, t_\beta \in_U Z_q$ and then computes $u_\beta = g_{1,\beta}^{s_\beta} g_{2,\beta}^{t_\beta}$ mod p and v_β =$h_{1,\beta}^{s_\beta} h_{2,\beta}^{t_\beta}$ mod p. Let $f_\beta = (u_\beta, v_\beta)$ and τ_β =(s_β, t_β).
  - R picks $u_{1-\beta} \in G$ and $v_{1-\beta} \in G$ uniformly at random. Let $f_{1-\beta}$ =$(u_{1-\beta}, v_{1-\beta})$.
  - R sends (f_0, f_1) to S;
- Upon receiving (f_0, f_1), parsing f_0 as (u_0, v_0) and f_1 as (u_1, v_1), S checks that $1 \neq u_i \in G$ and $1 \neq v_i \in G$ (i =0, 1), if any of the conditions are violated, S outputs $\perp$; Otherwise, S further checks the condition $v_\alpha \overset{?}{=} u_\alpha^{sk_\alpha}$ mod p.
  - If $v_\alpha \neq u_\alpha^{sk_\alpha}$ mod p, S sends b =0 to R and retries the channel setup procedure;
  - If $v_\alpha = u_\alpha^{sk_\alpha}$ mod p, S sends b =1 to R and continues the message transfer step below.

Message transfer: On input $m \in \{0,1\}$ and α, S computes $m \oplus \alpha$. Let c =$m \oplus \alpha$. S then sends c to R. Upon receiving a ciphertext c', R obtains m' by computing $c' \oplus \beta$.

4.2 The Proof of Security

Theorem. Assuming that the Decisional Diffie-Hellman problem is hard in G, the non-commitment protocol π depicted above realizes universally composable security in the presence of adaptive adversaries.

Proof. There are four cases defined in the following proof, depending on when the real world adversary $\mathcal{A}$ makes its first corruption request (and thus the proof is tedious):

- Case 1: the real world adversary $\mathcal{A}$ makes its first corruption request after a secure channel has been set up successfully;
- Case 2: the real world adversary $\mathcal{A}$ makes its first corruption request after the sender S has received R's first message;
- Case 3: the real world adversary $\mathcal{A}$ makes its first corruption request after S has generated its first message, but before S receives R's first message;

- Case 4: the real world adversary $\mathcal{A}$ makes its first corruption request before any messages are generated.

We show that in each case above there exists an ideal-world adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and players running π, or with $\mathcal{S}$ and $\mathcal{F}_{\rm SC}^{\mathcal{N}}$ in the ideal execution if the decisional Diffie-Hellman assumption holds.

To simplify the description of a simulator, we omit the explicit description of the non-information oracle $\mathcal{N}$ here and what follows since the non-commitment encryption scheme described in this paper is a well-structured protocol (informally, a well-structured protocol requires the message sizes and the number of rounds are completely determined by the protocol and are independent of the input values or random coins of the parties. For the details definition of well-structured protocol, please refer to (Garay et al., 2009)). We here and what follows, also omit the explicit checks that the simulator has seen the previous steps of the protocol.

Case 1. *The first corruption occurs after a secure channel has been set up successfully*. If the real world adversary $\mathcal{A}$ makes its first corruption request after a secure channel has been set up successfully, an ideal world adversary $\mathcal{S}$ must simulate any of the following three cases: 1) the first corruption occurs after R has received c; or 2) the first corruption occurs after S has generated c, but before R receives c; or 3) the first corruption occurs before S generates c. The corresponding simulator $\mathcal{S}$ is described as follows.

- Step 1: $\mathcal{S}$ first picks $g_{i,0} \in_U G$, $g_{i,1} \in_U G$, $sk_0 \in_U Z_q$ and $sk_1 \in_U Z_q$, and then computes $h_{i,0} = g_{i,0}^{sk_0} \bmod p$, $h_{i,1} = g_{i,1}^{sk_1} \bmod p$, i =1, 2. Let $e_0 = (g_{1,0}, g_{2,0}, h_{1,0}, h_{2,0})$ and $e_1 = (g_{1,1}, g_{2,1}, h_{1,1}, h_{2,1})$. $\mathcal{S}$ keeps the auxiliary strings sk_0 and sk_1 secret.
- Step 2: $\mathcal{S}$ then picks $s_i \in_U Z_q$ and $t_i \in_U Z_q$, and computes $u_i = g_{1,i}^{s_i} g_{2,i}^{t_i} \bmod p$, $v_i = h_{1,i}^{s_i} h_{2,i}^{t_i} \bmod p$. Let $f_i = (u_i, v_i)$, i =0, 1. $\mathcal{S}$ keeps the auxiliary strings (s_0, t_0) and (s_1, t_1) secret.
- Step 3: $\mathcal{S}$ outputs a bit b (=1).

Case 1.1. *the first corruption occurs after R has received c*; If a party $P \in \{S, R\}$ gets corrupted, $\mathcal{S}$ corrupts the corresponding dummy party $\widetilde{P}$ and obtains m. Let $\gamma = m \oplus c$. Following the steps (Step 1, Step 2 and Step 3) above, we further consider subcases below:

Case 1.1.1. If the sender S gets corrupted in the first corruption, $\mathcal{S}$ invokes the faking algorithm fake which takes S's internal state r_S as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled quadruple $r_{e_{1-\gamma}}$. That is, the faking algorithm fake interprets $g_{i,j} \in G$ as a string $r_{g_{i,j}} \in \{0,1\}^{2|p|}$ and $h_{i,j} \in G$ as a string $r_{h_{i,j}} \in \{0,1\}^{2|p|}$ (i =1, 2, j =0, 1). Let $r_{e_\gamma} = (r_{g_{1,\gamma}}, r_{g_{2,\gamma}}, r_{h_{1,\gamma}}, r_{h_{2,\gamma}})$ and $r_{e_{1-\gamma}} = (r_{g_{1,1-\gamma}}, r_{g_{2,1-\gamma}}, r_{h_{1,1-\gamma}}, r_{h_{2,1-\gamma}})$. The auxiliary string sk_γ is associated with r_{e_γ} such that $(r_{g_{1,\gamma}}, r_{h_{1,\gamma}}) \in \mathrm{Dlog}(sk_\gamma)$ and $(r_{g_{2,\gamma}}, r_{h_{2,\gamma}}) \in \mathrm{Dlog}(sk_\gamma)$ (*Note that given a string $r_{g_{i,j}} \in \{0,1\}^{2|p|}$, the corresponding group element $g_{i,j} \in G$ can be efficiently reconstructed by applying the Canetti-Fischlin's sampling algorithm*). $\mathcal{S}$ reveals (r_{e_0}, r_{e_1}), sk_γ and m to $\mathcal{A}$.

If R gets corrupted in the second corruption, $\mathcal{S}$ modifies the receiver's internal state r_R before it is revealed to $\mathcal{A}$. That is, $\mathcal{S}$ first invokes the faking algorithm fake which takes R's internal state r_R as input and interprets f_γ (=(u_γ, v_γ)) as a selection string r_{f_γ} (=$(r_{u_\gamma}, r_{v_\gamma})$) associated with the auxiliary string (s_γ, t_γ) and interpreted $f_{1-\gamma}$ (=$(u_{1-\gamma}, v_{1-\gamma})$) as a garbled string $r_{f_{1-\gamma}}$ (=$(r_{u_{1-\gamma}}, r_{v_{1-\gamma}})$). $\mathcal{S}$ reveals (r_{f_0}, r_{f_1}) and (s_γ, t_γ) to $\mathcal{A}$.

Case 1.1.2. When R gets corrupted in the first corruption, $\mathcal{S}$ corrupts the dummy party $\widetilde{R}$ in the ideal world and obtains m. $\mathcal{S}$ then invokes the faking algorithm fake which takes R's internal state r_R as input and interprets f_γ as a selection string r_{f_γ} associated with the auxiliary string (s_γ, t_γ) and interprets $f_{1-\gamma}$ as a garbled string $r_{f_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{f_0}, r_{f_1}) and (s_γ, t_γ) to $\mathcal{A}$.

If S gets corrupted in the second corruption, $\mathcal{S}$ corrupts the dummy party $\widetilde{S}$ in the ideal world and obtains m. $\mathcal{S}$ invokes the faking algorithm fake which takes r_S as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled string $r_{e_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{e_0}, r_{e_1}), sk_γ and m to $\mathcal{A}$.

Case 1.2. *The first corruption occurs after S has generated c, but before R receives c*; Following the steps (Step 1, Step 2 and Step 3) above, we further consider subcases below:

Case 1.2.1. If S gets corrupted in the first corruption after it has generated c, but before R receives c. $\mathcal{S}$ corrupts the dummy party $\widetilde{S}$ in the ideal world and obtains m. Let $\gamma = c \oplus m$. $\mathcal{S}$ invokes the faking algorithm fake which takes r_S as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled string $r_{e_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{e_0}, r_{e_1}), sk_γ and m to $\mathcal{A}$.

If the second corruption occurs *before* R receives a ciphertext c' (the received ciphertetxt c' may not be the same as the ciphertext c generated by S since the real-world adversary $\mathcal{A}$ may change the ciphertext c). $\mathcal{S}$ invokes the faking algorithm fake which takes r_R as input and interprets f_γ as a selection string r_{f_γ} associated with the auxiliary string (s_γ, t_γ) and interprets $f_{1-\gamma}$ as a garbled string $r_{f_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{f_0}, r_{f_1}) and (s_γ, t_γ) to $\mathcal{A}$.

If the second corruption occurs *after* R has received a ciphertext c'. The simulator invokes fake which takes r_R as input and interprets f_γ as a selection string associated with the auxiliary string (s_γ, t_γ) and interprets $f_{1-\gamma}$ as a garbled string. Let $m' = \gamma \oplus c'$. $\mathcal{S}$ reveals (r_{f_0}, r_{f_1}), (s_γ, t_γ) and m' to $\mathcal{A}$.

Case 1.2.2. If the receiver R gets corrupted in the first corruption, $\mathcal{S}$ corrupts the corresponding dummy party $\widetilde{R}$ and obtains β. Let $\gamma = \beta$. $\mathcal{S}$ invokes the faking algorithm fake which takes R's internal state r_R as input and interprets f_γ as a selection string r_{f_γ} associated with the auxiliary string (s_γ, t_γ) and interprets $f_{1-\gamma}$ as a garbled string $r_{f_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{f_0}, r_{f_1}) and (s_γ, t_γ) to $\mathcal{A}$.

If the second corruption occurs, $\mathcal{S}$ corrupts the corresponding dummy party $\widetilde{S}$ and obtains m. $\mathcal{S}$ then invokes the faking algorithm fake which takes r_S as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled string $r_{e_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{e_0}, r_{e_1}), sk_γ and m to $\mathcal{A}$.

Case 1.3. *The first corruption occurs before S generates c*; Following the simulation steps (Step 1, Step 2 and Step 3) above, we consider the following two subcases:

Case 1.3.1. If the sender S gets corrupted in the first corruption, $\mathcal{S}$ corrupts the corresponding dummy party $\widetilde{S}$ in the ideal world and obtains m. $\mathcal{S}$ picks a random bit $\gamma \in \{0,1\}$ uniformly at random. Let $\alpha = \gamma$ and $\beta = \gamma$. $\mathcal{S}$ invokes the faking algorithm fake which takes r_S as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled quadruple $r_{e_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{e_0}, r_{e_1}), sk_γ and m to $\mathcal{A}$.

If R gets corrupted in the second corruption, $\mathcal{S}$ invokes the faking algorithm fake which takes R's internal state r_R as input and interprets f_γ as a selection string r_{f_γ} associated with the auxiliary string (s_γ, t_γ) and interprets $f_{1-\gamma}$ as a garbled string $r_{f_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{f_0}, r_{f_1}) and (s_γ, t_γ) to $\mathcal{A}$.

Case 1.3.2. If the receiver R gets corrupted in the first corruption, $\mathcal{S}$ picks a bit $\gamma \in \{0,1\}$ uniformly at random and then invokes the faking algorithm fake which takes r_R as input and interprets f_γ as a selection string r_{f_γ} associated with the auxiliary string (s_γ, t_γ) and interprets $f_{1-\gamma}$ as a garbled string $r_{f_{1-\gamma}}$. $\mathcal{S}$ reveals(r_{f_0}, r_{f_1}) and (s_γ, t_γ) to $\mathcal{A}$.

If S gets corrupted in the second corruption, $\mathcal{S}$ corrupts the corresponding dummy party $\widetilde{S}$ and obtains m. $\mathcal{S}$ invokes the faking algorithm fake which takes r_S as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled quadruple $r_{e_{1-\gamma}}$. The simulator reveals (r_{e_0}, r_{e_1}), sk_γ and m to $\mathcal{A}$.

Case 2. *The first corruption occurs after the sender S has received R's first message.* If the real world adversary $\mathcal{A}$ makes its first corruption after the sender S has received R's first message (f_0, f_1), the constructed ideal world adversary $\mathcal{S}$ must simulate any of the following three subcases: 1) the first corruption occurs after S has generated b and R has received b; or 2) the first corruption occurs after S has generated b, but before R receives b; or 3) the first corruption occurs before S generates b. We describe the corresponding simulator $\mathcal{S}$ below

- Step 1: $\mathcal{S}$ picks $g_{i,0} \in_U G$, $g_{i,1} \in_U G$, $sk_0 \in_U Z_q$ and $sk_1 \in_U Z_q$, and then computes $h_{i,0} = g_{i,0}^{sk_0} \bmod p$, $h_{i,1} = g_{i,1}^{sk_1} \bmod p$. Let $e_i = (g_{1,i}, g_{2,i}, h_{1,i}, h_{2,i})$, $i = 1, 2$. $\mathcal{S}$ keeps the auxiliary strings sk_0 and sk_1 secret.
- Step 2: $\mathcal{S}$ picks $s_i \in Z_q$ and $t_i \in Z_q$ uniformly at random, and then computes $u_i = g_{1,i}^{s_i} g_{2,i}^{t_i} \bmod p$, $v_i = h_{1,i}^{s_i} h_{2,i}^{t_i} \bmod p$. Let $f_i = (u_i, v_i)$, $i = 0, 1$. $\mathcal{S}$ keeps the auxiliary strings (s_0, t_0) and (s_1, t_1) secret.

Case 2.1. *The first corruption occurs after the sender S has received (f_0, f_1) and R has received a bit b.* Following the simulation steps (Step 1 and Step 2) above, we further consider subcases below:

Case 2.1.1. If b=1 and a party $P \in \{S, R\}$ gets corrupted in the first corruption, the corresponding simulator can be constructed exactly as that described in Case 1.

Case 2.1.2. If b=0 and if the sender S gets corrupted in the first corruption, the simulator $\mathcal{S}$ corrupts the corresponding dummy party $\widetilde{S}$ in the ideal world and obtains m. $\mathcal{S}$ then chooses a random bit $\gamma \in_U \{0,1\}$. Let $\alpha = \gamma$ and $\beta = 1 - \gamma$. $\mathcal{S}$ invokes the faking algorithm fake which takes r_S as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled quadruple $r_{e_{1-\gamma}}$ and reveals (r_{e_0}, r_{e_1}), sk_γ and m to the real world adversary $\mathcal{A}$.

If the receiver R gets corrupted in the second corruption, $\mathcal{S}$ invokes fake which takes r_R as input and interprets $f_{1-\gamma}$ as a selection string $r_{f_{1-\gamma}}$ associated with the auxiliary string $(s_{1-\gamma}, t_{1-\gamma})$ and interprets f_γ as a garbled string r_{f_γ} and reveals (r_{f_0}, r_{f_1}) and $(s_{1-\gamma}, t_{1-\gamma})$ to $\mathcal{A}$.

Case 2.1.3. If b=0 and if the receiver R gets corrupted in the first corruption, $\mathcal{S}$ picks a bit $\gamma \in \{0,1\}$ uniformly at random and sets $\beta = \gamma$ and $\alpha = 1-\gamma$. $\mathcal{S}$ invokes the faking algorithm fake which takes r_R as input and interprets f_γ as a selection string r_{f_γ} associated with the auxiliary string (s_γ, t_γ) and interprets $f_{1-\gamma}$ as a garbled string $r_{f_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{f_0}, r_{f_1}) and (s_γ, t_γ) to $\mathcal{A}$.

If the sender S gets corrupted in the second corruption, the simulator $\mathcal{S}$ corrupts the corresponding dummy party $\widetilde{S}$ in the ideal world and obtains m. $\mathcal{S}$ then interprets $e_{1-\gamma}$ as a Diffie-Hellman quadruple $r_{e_{1-\gamma}}$ associated with the auxiliary string $sk_{1-\gamma}$ and interprets e_γ as a garbled quadruple r_{e_γ}. $\mathcal{S}$ then reveals (r_{e_0}, r_{e_1}), $sk_{1-\gamma}$ and m to $\mathcal{A}$.

Case 2.2. *The first corruption occurs after S has received* (f_0, f_1) *and S has generated b but before R receives the bit b.* Following the simulation steps (Step 1 and Step 2) above, we further consider subcases below:

Case 2.2.1. If S gets corrupted in the first corruption and if b =1, $\mathcal{S}$ corrupts a dummy party $\widetilde{S}$ and obtains m. $\mathcal{S}$ then picks a bit $\gamma \in_U \{0,1\}$. Let $\alpha = \gamma$ and $\beta = \gamma$. $\mathcal{S}$ invokes the faking algorithm fake which takes r_S as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled quadruple $r_{e_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{e_0}, r_{e_1}), sk_γ and m to $\mathcal{A}$.

If the receiver R gets corrupted before R receives the bit b (=1) in the second corruption, $\mathcal{S}$ invokes fake algorithm which takes r_R as input and interprets f_γ as a selection string r_{f_γ} associated with the auxiliary string (s_γ, t_γ) and interprets $f_{1-\gamma}$ as a garbled string $r_{f_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{f_0}, r_{f_1}) and (s_γ, t_γ) to $\mathcal{A}$. If the receiver R gets corrupted after it has received a bit b' (the generated bit might be changed by $\mathcal{A}$) in the second corruption, the corresponding simulator can be constructed exactly as that described in Case 2.1.

Case 2.2.2. If R gets corrupted in the first corruption and if b=1, $\mathcal{S}$ picks a random bit $\gamma \in_U \{0,1\}$ and sets $\beta = \gamma$ and $\alpha = \gamma$. $\mathcal{S}$ interprets f_γ as a selection string r_{f_γ} associated with the auxiliary string (s_γ, t_γ) and interprets $f_{1-\gamma}$ as a garbled string $r_{f_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{f_0}, r_{f_1}) and (s_γ, t_γ) to $\mathcal{A}$

If S gets corrupted in the second corruption, $\mathcal{S}$ corrupts the dummy party $\widetilde{S}$ and obtains m. $\mathcal{S}$ then interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled quadruple $r_{e_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{e_0}, r_{e_1}), sk_γ and m to $\mathcal{A}$.

Case 2.2.3. If S gets corrupted in the first corruption and if b =0, $\mathcal{S}$ corrupts a dummy party S and obtains m and picks a bit $\gamma \in \{0,1\}$ uniformly at random. Let $\alpha = \gamma$ and $\beta = 1-\gamma$. $\mathcal{S}$ invokes the faking algorithm fake which takes r_S as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled quadruple $r_{e_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{e_0}, r_{e_1}), sk_γ and m to $\mathcal{A}$.

If the receiver R gets corrupted before it receives a bit b' in the second corruption, $\mathcal{S}$ interprets $f_{1-\gamma}$ a selection string $r_{f_{1-\gamma}}$ associated with the auxiliary string $(s_{1-\gamma}, t_{1-\gamma})$ and interprets f_γ as a garbled string r_{f_γ}. $\mathcal{S}$ reveals the randomness (r_{f_0}, r_{f_1}), $(s_{1-\gamma}, t_{1-\gamma})$ to $\mathcal{A}$. If the receiver R gets corrupted after it has received a bit b', the simulator can be constructed exactly as that described in Case 2.1.

Case 2.2.4. If R gets corrupted in the first corruption and if b=0, $\mathcal{S}$ picks a random bit $\gamma \in \{0,1\}$ uniformly at random and sets $\beta = \gamma$ and $\alpha = 1-\gamma$. $\mathcal{S}$ interprets f_γ as a selection string r_{f_γ} associated with the auxiliary string sk_γ and interprets $f_{1-\gamma}$ as a garbled string $r_{f_{1-\gamma}}$. $\mathcal{S}$ then reveals (r_{f_0}, r_{f_1}), (s_γ, t_γ) to $\mathcal{A}$. If S gets corrupted in the second corruption, $\mathcal{S}$ corrupts $\widetilde{S}$ and obtains m, and then interprets $e_{1-\gamma}$ as a Diffie-Hellman quadruple $r_{e_{1-\gamma}}$ associated with the auxiliary string $sk_{1-\gamma}$ and interprets e_γ as a garbled quadruple r_{e_γ}. $\mathcal{S}$ reveals (r_{e_0}, r_{e_1}), $sk_{1-\gamma}$ and m to $\mathcal{A}$.

Case 2.3. *The first corruption occurs after the sender S has received* (f_0, f_1)*, but before S generates b.* Following the simulation steps (Step 1 and Step 2) above, $\mathcal{S}$ picks a bit $b \in \{0,1\}$ uniformly at random. We further consider the following subcases

Case 2.3.1. If S gets corrupted in the first corruption and if b =1, the corresponding simulator $\mathcal{S}$ can be constructed exactly as that described in Case 2.2.1;

Case 2.3.2. If R gets corrupted in the first corruption and if b =1, the corresponding simulator $\mathcal{S}$ can be constructed exactly as that described in Case 2.2.2;

Case 2.3.3. If S gets corrupted in the first corruption and if $b = 0$,the corresponding simulator $\mathcal{S}$ can be constructed exactly as that described in Case 2.2.3;

Case 2.3.4. If R gets corrupted in the first corruption and if $b = 0$, the corresponding simulator $\mathcal{S}$ can be constructed exactly as that described in Case 2.2.4.

Case 3. *The first corruption occurs after S has generated its first message, but before it receives R's first message.* If the real world adversary $\mathcal{A}$ makes its first corruption request after S has generated its first message, but before it receives R's first message, an ideal world adversary $\mathcal{S}$ must simulate any of the following subcases: 1) the first corruption occurs after R has received (e_0, e_1) and have generated its first message (f_0, f_1) but before S receives it; or 2) the first corruption occurs after R has received (e_0, e_1) but before R generates (f_0, f_1); or 3) the first corruption occurs after S has generated its first message, but before R

receives S' first message.

Case 3.1. *If the first corruption occurs after R has received (e_0,e_1) and R has generated (f_0,f_1) but before S receives it.* The corresponding simulator $\mathcal{S}$ can be constructed as follows.

- Step 1: $\mathcal{S}$ picks $g_{i,0} \in_U G$, $g_{i,1} \in_U G$, $sk_0 \in_U Z_q$ and $sk_1 \in_U Z_q$, and then computes $h_{i,0} = g_{i,0}^{sk_0} \bmod p$, $h_{i,1} = g_{i,1}^{sk_1} \bmod p$, i =1, 2. Let $e_0 = (g_{1,0},g_{2,0},h_{1,0},h_{2,0})$ and $e_1 = (g_{1,1},g_{2,1},h_{1,1},h_{2,1})$. $\mathcal{S}$ keeps the auxiliary strings sk_0 and sk_1 secret.
- Step 2: $\mathcal{S}$ then picks $s_i \in_U Z_q$ and $t_i \in_U Z_q$, and then computes $u_i = g_{1,i}^{s_i} g_{2,i}^{t_i} \bmod p$, $v_i = h_{1,i}^{s_i} h_{2,i}^{t_i} \bmod p$. Let $f_i = (u_i,v_i)$, i =0, 1. $\mathcal{S}$ keeps the auxiliary strings (s_0,t_0) and (s_1,t_1) secret.

Following the simulation steps (Step 1 and Step 2) above, we consider the following subcases:

Case 3.1.1. If S gets corrupted in the first corruption, $\mathcal{S}$ corrupts the corresponding dummy party $\widetilde{S}$ in the ideal world and obtains m. $\mathcal{S}$ then picks a bit $\gamma \in_U \{0,1\}$ uniformly at random and sets $\alpha = \gamma$; $\mathcal{S}$ invokes the faking algorithm fake which takes r_S as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled quadruple $r_{e_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{e_0}, r_{e_1}), sk_γ and m to $\mathcal{A}$.

If R gets corrupted in the second corruption, $\mathcal{S}$ chooses a bit $b \in_U \{0,1\}$ uniformly at random. We further consider the following cases

- if $b = 1$, then let $\beta = \gamma$. $\mathcal{S}$ invokes the faking algorithm fake which takes r_R as input and interprets f_γ as a selection string r_{f_γ} associated with the auxiliary string (s_γ,t_γ) and interprets $f_{1-\gamma}$ as a garbled string $r_{f_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{f_0}, r_{f_1}) and (s_γ,t_γ) to $\mathcal{A}$.
- if b =0, then let $\beta = 1-\gamma$. $\mathcal{S}$ invokes the faking algorithm fake which takes r_R as input and interprets $f_{1-\gamma}$ as a selection string $r_{f_{1-\gamma}}$ associated with the auxiliary string $(s_{1-\gamma},t_{1-\gamma})$ and interprets f_γ as a garbled string r_{f_γ}. $\mathcal{S}$ reveals (r_{f_0}, r_{f_1}) and $(s_{1-\gamma},t_{1-\gamma})$ to $\mathcal{A}$.

Case 3.1.2. If R gets corrupted in the first corruption, $\mathcal{S}$ picks a bit $\gamma \in_U \{0,1\}$ uniformly at random and sets $\beta = \gamma$; $\mathcal{S}$ invokes the faking algorithm fake which takes r_R as input and interprets f_γ as a selection string r_{f_γ} associated with the auxiliary string (s_γ,t_γ) and interprets $f_{1-\gamma}$ as a garbled string $r_{f_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{f_0}, r_{f_1}) and (s_γ,t_γ) to $\mathcal{A}$.

If the sender S gets corrupted in the second corruption, $\mathcal{S}$ chooses a bit $b \in_U \{0,1\}$ uniformly at random. We further consider the following cases

- if $b = 1$, let $\beta = \gamma$. $\mathcal{S}$ invokes the faking algorithm fake which takes r_R as input and interprets f_γ as a selection string r_{f_γ} associated with the auxiliary string (s_γ,t_γ) and interprets $f_{1-\gamma}$ as a garbled string $r_{f_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{f_0}, r_{f_1}) and (s_γ,t_γ) to $\mathcal{A}$.
- if b =0, let $\beta = 1-\gamma$. $\mathcal{S}$ invokes the faking algorithm fake which takes r_R as input and interprets $f_{1-\gamma}$ as a selection string $r_{f_{1-\gamma}}$ associated with the auxiliary string $(s_{1-\gamma},t_{1-\gamma})$ and interprets f_γ as a garbled string r_{f_γ}. $\mathcal{S}$ reveals (r_{f_0}, r_{f_1}) and $(s_{1-\gamma},t_{1-\gamma})$ to $\mathcal{A}$.

Case 3.2. If the first corruption occurs after R has received (e_0,e_1) but before R generates (f_0,f_1), the corresponding simulator $\mathcal{S}$ can be constructed as follows.

- Step 1: $\mathcal{S}$ picks $g_{i,0} \in_U G$, $g_{i,1} \in_U G$, $sk_0 \in_U Z_q$ and $sk_1 \in_U Z_q$ uniformly at random, and then computes $h_{i,0} = g_{i,0}^{sk_0} \bmod p$, $h_{i,1} = g_{i,1}^{sk_1} \bmod p$, i =1, 2. Let $e_0 = (g_{1,0},g_{2,0},h_{1,0},h_{2,0})$ and $e_1 = (g_{1,1},g_{2,1},h_{1,1},h_{2,1})$. $\mathcal{S}$ keeps the auxiliary strings sk_0 and sk_1 secret.

Following the simulation Step 1 above, we further consider subcases below:

Case 3.2.1. If the sender S gets corrupted in the first corruption, $\mathcal{S}$ corrupts the corresponding dummy party $\widetilde{S}$ in the ideal world and obtains m. $\mathcal{S}$ then picks a bit $\gamma \in \{0,1\}$ uniformly at random and sets $\alpha = \gamma$; $\mathcal{S}$ invokes the faking algorithm fake which takes r_S as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled quadruple $r_{e_{1-\gamma}}$. $\mathcal{S}$ reveals m, (r_{e_0}, r_{e_1}) and sk_γ to $\mathcal{A}$.

- if R gets corrupted before it generates (f_0,f_1) in the second corruption, $\mathcal{S}$ reveals R's internal state r_R to $\mathcal{A}$; The rest of simulation is trivial since both parties have already corrupted.
- if R gets corrupted after it has generated (f_0,f_1) in the second corruption, $\mathcal{S}$ picks a bit $b \in \{0,1\}$ uniformly at random

  1) if $b = 1$, $\mathcal{S}$ invokes the faking algorithm fake which takes r_R as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled quadruple $r_{e_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{e_0}, r_{e_1}) and sk_γ to $\mathcal{A}$.

  2) if $b = 0$, $\mathcal{S}$ invokes the faking algorithm fake which takes r_R as input and interprets $e_{1-\gamma}$ as a Diffie-Hellman quadruple $r_{e_{1-\gamma}}$ associated with the auxiliary string $sk_{1-\gamma}$ and interprets e_γ as a garbled quadruple r_{e_γ}. $\mathcal{S}$ reveals (r_{e_0}, r_{e_1}) and $sk_{1-\gamma}$ to $\mathcal{A}$.

Case 3.2.2. If the receiver R gets corrupted in the first corruption, $\mathcal{S}$ simply reveals r_R to $\mathcal{A}$.

- if S gets corrupted before it obtains (f_0', f_1') in the second corruption, $\mathcal{S}$ corrupts the corresponding dummy party $\widetilde{S}$ in the ideal world and obtains m. $\mathcal{S}$ picks a bit $\gamma \in \{0,1\}$ uniformly at random and sets $\alpha = \gamma$. $\mathcal{S}$ invokes the faking algorithm fake which takes r_S as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled quadruple $r_{e_{1-\gamma}}$. $\mathcal{S}$ reveals m, (r_{e_0}, r_{e_1}) and sk_γ to $\mathcal{A}$.
- if S gets corrupted after it has received (f_0', f_1') in the second corruption. We consider the following two cases:

  1) if (f_0', f_1') is not well-defined, i.e., any of the conditions $1 \neq u_i \in G$ and $1 \neq v_i \in G$ (i =0, 1) are violated, $\mathcal{S}$ picks a bit $b \in \{0,1\}$ uniformly at random. If $b = 1$, let $\alpha = \gamma$; if b =0, let $\alpha = 1-\gamma$. The rest work of simulator is same as that described in Case 2.1.

  2) if (f_0', f_1') is well-defined, i.e., $1 \neq u_i \in G$ and $1 \neq v_i \in G$ (i =0, 1), $\mathcal{S}$ picks a bit $b \in \{0,1\}$ uniformly at random. If $b = 1$, let $\beta = \gamma$; If $b = 0$, let $\beta = 1-\gamma$. The rest work of simulator is same as that described in Case 2.1.

Case 3.3. If the first corruption occurs after S has generated (e_0, e_1) but before R receives it, the corresponding simulator $\mathcal{S}$ can be constructed as follows.

- Step 1: $\mathcal{S}$ picks $g_{i,0} \in_U G$, $g_{i,1} \in_U G$, $sk_0 \in_U Z_q$ and $sk_1 \in_U Z_q$, and then computes $h_{i,0} = g_{i,0}^{sk_0} \bmod p$, $h_{i,1} = g_{i,1}^{sk_1} \bmod p$, i =1, 2. Let $e_0 = (g_{1,0}, g_{2,0}, h_{1,0}, h_{2,0})$ and $e_1 = (g_{1,1}, g_{2,1}, h_{1,1}, h_{2,1})$. $\mathcal{S}$ keeps the auxiliary strings sk_0 and sk_1 secret.

Following the simulation Step 1 above, we further consider subcases below:

Case 3.3.1. If S gets corrupted in the first corruption, $\mathcal{S}$ picks a bit $\gamma \in \{0,1\}$ uniformly at random and sets $\alpha = \gamma$; $\mathcal{S}$ invokes the faking algorithm fake which takes r_S as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled quadruple $r_{e_{1-\gamma}}$. $\mathcal{S}$ reveals m, (r_{e_0}, r_{e_1}) and sk_γ to $\mathcal{A}$.

- if R gets corrupted before it receives (e_0', e_1') in the second corruption, $\mathcal{S}$ reveals R's internal state r_R to $\mathcal{A}$. The rest of simulation is trivial since both parties have already got corrupted.
- if R gets corrupted after it has received (e_0', e_1'). If (e_0', e_1') is not well-defined, i.e., any of the conditions $1 \neq g_{i,j} \in G$ and $1 \neq h_{i,j} \in G$ (i =1, 2, j =0, 1) are violated, then $\mathcal{S}$ reveal r_R to $\mathcal{A}$; If (e_0', e_1') is well-defined, i.e., $1 \neq g_{i,j} \in G$ and $1 \neq h_{i,j} \in G$ (i =1, 2, j =0, 1), $\mathcal{S}$ picks a random bit $\gamma \in \{0,1\}$ and sets $\alpha = \gamma$. The rest work of $\mathcal{S}$ is same as that described in Case 3.2.

Case 3.3.2. If R gets corrupted in the first corruption, $\mathcal{S}$ reveals R's internal state r_R to $\mathcal{A}$. If S gets corrupted in the second corruption, we consider the following two cases:

- if (f_0', f_1') has not been received, $\mathcal{S}$ picks a random bit $\gamma \in_U \{0,1\}$ and sets $\alpha = \gamma$. $\mathcal{S}$ then invokes the faking algorithm fake which takes r_S as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled quadruple. $\mathcal{S}$ reveals m, (r_{e_0}, r_{e_1}) and sk_γ to the real world adversary $\mathcal{A}$.
- if (f_0', f_1') has been received, and if (f_0', f_1') is not well-defined, the rest work of $\mathcal{S}$ is same as that described in Case 3.2; if (f_0', f_1') has been received and if (f_0', f_1') is well-defined, $\mathcal{S}$ can be constructed exactly as that described in Case 2.3.

Case 4. *The first corruption occurs before* (e_0, e_1) *has been generated*. If $\mathcal{A}$ makes its first request before S generates (e_0, e_1), the corresponding simulator $\mathcal{S}$ can be constructed as follows.

Case 4.1. If S gets corrupted in the first corruption, $\mathcal{S}$ corrupts the corresponding dummy party $\widetilde{S}$ in the ideal world and obtains m. $\mathcal{S}$ reveals its internal state r_S together with its input m to $\mathcal{A}$.

- if R gets corrupted before R generates (f_0, f_1) in the second corruption, $\mathcal{S}$ reveals R's internal state r_R to $\mathcal{A}$.
- if R gets corrupted after R has generated (f_0, f_1) in the second corruption, $\mathcal{S}$ picks a bit $\gamma \in_U \{0,1\}$ uniformly at random and sets $\beta = \gamma$. $\mathcal{S}$ invokes the faking algorithm fake which takes r_R as input and interprets f_γ as a selection string r_{f_γ} associated with the auxiliary string (s_γ, t_γ) and interprets $f_{1-\gamma}$ as a garbled string $r_{f_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{f_0}, r_{f_1}) and (s_γ, t_γ) to $\mathcal{A}$.

Case 4.2. If R gets corrupted in the first corruption, the corresponding simulator $\mathcal{S}$ can be constructed as follows.

- Step 1: $\mathcal{S}$ picks $g_{i,0} \in_U G$, $g_{i,1} \in_U G$, $sk_0 \in_U Z_q$ and $sk_1 \in_U Z_q$, and then computes $h_{i,0} = g_{i,0}^{sk_0} \bmod p$, $h_{i,1} = g_{i,1}^{sk_1} \bmod p$, i =1, 2. Let $e_0 = (g_{1,0}, g_{2,0}, h_{1,0}, h_{2,0})$ and $e_1 = (g_{1,1}, g_{2,1}, h_{1,1}, h_{2,1})$. $\mathcal{S}$ keeps the auxiliary strings sk_0 and sk_1 secret.

Following Step 1 above, $\mathcal{S}$ corrupts the corresponding dummy party $\widetilde{R}$ in the ideal world and reveals R's internal state r_R to $\mathcal{A}$. If S gets corrupted in the second corruption, we further consider subcases below:

- if S gets corrupted before R generates (f'_0, f'_1), or if S gets corrupted after R has generated (f'_0, f'_1), but before S receives (f'_0, f'_1), $\mathcal{S}$ corrupts the corresponding dummy party $\widetilde{S}$ in the ideal world and obtains m. $\mathcal{S}$ picks a random bit $\gamma \in \{0,1\}$ uniformly at random and sets $\alpha = \gamma$. $\mathcal{S}$ invokes the faking algorithm fake which takes r_S as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled quadruple $r_{e_{1-\gamma}}$. $\mathcal{S}$ reveals (r_{e_0}, r_{e_1}), sk_γ and m to $\mathcal{A}$.
- if S gets corrupted after S has received (f'_0, f'_1), we further consider the following two cases:

  - 1) if (f'_0, f'_1) is not well-defined, $\mathcal{S}$ picks a random bit $\gamma \in_U \{0,1\}$ and sets $\alpha = \gamma$. $\mathcal{S}$ then invokes the faking algorithm fake which takes r_S as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled quadruple $r_{e_{1-\gamma}}$. $\mathcal{S}$ reveals m, (r_{e_0}, r_{e_1}) and sk_γ to $\mathcal{A}$.

  - 2) if (f'_0, f'_1) is well-defined, $\mathcal{S}$ further checks $v'_i \overset{?}{=} {u'_i}^{sk_i}$ for i =0, 1. If both indices are invalid, $\mathcal{S}$ does the same procedure in the above case and reveals m, (r_{e_0}, r_{e_1}) and sk_γ to $\mathcal{A}$. If there exists an index i satisfied with the check condition, $\mathcal{S}$ picks a bit $b \in \{0,1\}$ uniformly at random. If b=1, then let $\alpha = i$. $\mathcal{S}$ invokes the faking algorithm fake which takes r_S as input and interprets e_γ as a Diffie-Hellman quadruple r_{e_γ} associated with the auxiliary string sk_γ and interprets $e_{1-\gamma}$ as a garbled quadruple $r_{e_{1-\gamma}}$. $\mathcal{S}$ reveals m, (r_{e_0}, r_{e_1}) and sk_γ to $\mathcal{A}$. If $b = 0$, let $\alpha = 1 - i$. $\mathcal{S}$ invokes the faking algorithm fake which takes r_S as input and interprets $e_{1-\gamma}$ as a random Diffie-Hellman quadruple $r_{e_{1-\gamma}}$ associated with the auxiliary string $sk_{1-\gamma}$ and interprets e_γ as a garbled quadruple. $\mathcal{S}$ reveals m, (r_{e_0}, r_{e_1}) and $sk_{1-\gamma}$ to $\mathcal{A}$.

By the DDH assumption, we know that the distribution of random variable e_γ is computationally indistinguishable from that of $e_{1-\gamma}$. Due to the randomness of Naor-Pinkas randomizer, the distribution of random variable f_γ is computationally indistinguishable from that of $f_{1-\gamma}$. This means that $\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable in all cases. As a result, the real-world protocol π realizes $\mathcal{F}_{\mathrm{SC}}^{\mathcal{N}}$.

5 CONCLUSIONS

In this paper, a new implementation of non-committing encryptions has been presented and analyzed. We have shown that the proposed non-committing encryption scheme realizes the UC-security in the presence of adaptive adversary assuming that the decisional Diffie-Hellman problem is hard.

REFERENCES

Beaver, D. (1997). Plug and play encryption. In *CRYPTO*. Springer.

Beaver, D. and Haber, S. (1992). Cryptographic protocols provably secure against dynamic adversaries. In *EUROCRYPT*. Springer.

Canetti, R. (2001). a new paradigm for cryptographic protocols. In *FOC*. IEEE.

Canetti, R. (2005). Universally composable security: A new paradigm for cryptographic protocols. In *ePrint*. eprinter.iacr.org.

Canetti, R., Feige, U., Goldreich, O., and Naor, M. (1996). Adaptively secure multi-party computation. In *STOC*. IEEE.

Canetti, R. and Fischlin, M. (2001). a new paradigm for cryptographic protocols. In *CRYPTO*. Springer.

Damgård, I. and Nielsen, J. (2000). Improved non-committing encryption schemes based on a general complexity assumption. In *CRYPTO*. Springer.

Garay, J., Wichs, D., and Zhou, H. (2009). Somewhat non-committing encryption and efficient adaptively secure oblivious transfer. In *CRYPTO*. Springer.

Moni Naor, B. P. (2001). Efficient oblivious transfer protocols. In *SODA*. ACM.

Nielsen, J. (2002). Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *CRYPTO*. Springer.

Nielsen, J. (2003). On protocol security in the cryptographic model. In *thesis*. www.brics.dk/ jbn/thesis.pdf.

S.Choi, Dachman-Soled, D., Malkin, T., and Wee, H. (2009). Adaptively secure multi-party computation. In *Asiacrypt*. Springer.