

The Non-interactive Equivocable Non-malleable Commitment and its Applications

Chunming Tang Zhuojun Liu ¹⁾

Abstract. Under the assumption of the existence of one-way functions, we prove that it is possible to construct the non-interactive equivocable non-malleable commitment scheme. Especially importantly, we succeed in implementing concurrent zero-knowledge using our commitment scheme, analysing properties of this concurrent zero-knowledge proof system, comparing it with the existed concurrent zero-knowledge, and finding that our results are superior to their.

1. INTRODUCTION

Commitment scheme: Commitment scheme is a basic ingredient in many cryptographic protocols. It usually involves two probabilistic polynomial-time players: the committer and the receiver. Very informally, it consists of two phases, a commitment phase and a de-commitment phase. In the commitment phase, the committer with a secret input x engages in a protocol, receiver still does not know what x is (i.e., x is computationally hidden), at the same time, the committer can subsequently (i.e., during the de-commitment phase) open only one possible value of x . Commitment scheme can be divided in two types, according to whether the security property holds with respect to computationally bounded adversaries (i.e., computationally secure bit-commitment scheme) or to unbounded adversaries (i.e., perfectly-secure bit-commitment scheme).

Non-interactive commitment: Informally speaking, a bit commitment scheme (A, B) in the public-random-string model (i.e., non-interactive commitment) is a two-phase interactive protocol between two probabilistic polynomial time parties A (the committer) and B (the receiver) such that the following is true. In the first phase (the commitment phase), A commits to bit b by computing a pair of keys (com, dec) and sending com (the commitment key) to B . Given just the public random string and the commitment key, the polynomial-time receiver B can not guess the bit with probability significantly better than $1/2$ (this is the security property). In the second phase (the decommitment phase) A reveals the bit b and the key dec (the decommitment key) to B . Now B checks whether the decommitment key is valid; if not, B outputs a special string \perp , meaning that he rejects the decommitment from A ; otherwise, B can efficiently compute the bit b revealed by A and is convinced that b was indeed chosen by A in the first phase (this is the binding property).

Equivocable commitment: A bit commitment scheme is equivocable if it satisfies the following additional requirement. There exists an efficient simulator which outputs a transcript leading to a faked commitment such that: (a) the commitment can be decommitted both as 0 and as 1, and (b) the simulated transcript is indistinguishable from a real execution.

¹⁾Institute of Systems Science, Chinese Academy of Sciences, Beijing(100080)

Non-malleable commitment: The notion of non-malleable commitment can be best explained with the following motivating example from [1]: suppose there are several players who participate in a contract bidding game, where a contract goes to the lowest bidder. First, the players send the commitments of the bids, and once all bids have been deposited, they de-commit. In [1] it was observed that even if the commitment scheme is computationally secure against any polynomially-bounded receivers, still a malicious committer can potentially come up with a commitment of a related bid, without any knowledge what the original bid is, but still being able to underbid. The reason is that the standard notion of commitment does not disallow the ability to come up with the related commitments (for which an attacker does not know the de-commitment at the commitment stage), but for which once the attacker gets the de-commitment of the original value, he can compute the de-commitment of his related de-commitment as well.

Previous work: In [2, 3], the common random string model for non-interactive zero-knowledge proofs was introduced, a model where a polynomial-length common random string is available to all the users. *Di Crescenzo* and *Ostrovsky* in [4] constructed the interactive computationally equivocable and computationally secure commitment scheme if the existence of pseudo-random generators and perfectly secure commitment schemes. Furthermore, assuming the existence of computationally secure commitment schemes and the intractability of computing discrete logarithms modulo integers of the form $p = 2q + 1$, for p, q primes, they could obtain the perfectly equivocable and perfectly secure commitment scheme. Especially importantly, they used this two commitment scheme to succeed in implementing *concurrent* zero-knowledge proofs and arguments. The non-malleable commitment was first advanced by *Dolev, Dwork, Naor* in [1], subsequently, *De Santis* and *Persiano* [5], *Bellare* and *Rogaway* [6] also presented the non-malleable commitment, respectively. However, all the previous proposed solutions to this fundamental problem required either very strong assumptions or logarithmic number of rounds of interaction, and relied on inefficient zero-knowledge proofs. In [7], *Di Crescenzo, Ishai* and *Ostrovsky* formalized and implemented the non-interactive non-malleable commitment, only under the assumption of the existence of one-way functions.

In summary, the existed commitment schemes satisfy at most two of the three properties (i.e, non-interaction, equivocability and non-malleability), hence, looking for a commitment scheme satisfying all properties becomes our main object.

Our works: Our main results are as followed: (1) Under the assumption of the existence of one-way functions, we construct the commitment scheme satisfying non-interaction, equivocability and non-malleability. (2) Because of its strong robustness, we succeed in implementing the *concurrent* zero-knowledge proof with our commitment scheme, analysing properties of our concurrent zero-knowledge proof system model, and comparing it with the *concurrent* zero-knowledge proof system [4]. We find our results are superior to their results.

2. DEFINITIONS AND LEMMAS

In this section we recall some definitions and some lemmas.

2.1 BASIC NOTATIONS AND DEFINITIONS

Basic notations. The notation $x \leftarrow S$ denotes the random process of selecting element x from set S with uniform probability distribution over S . Similarly, if D is a distribution, the notation $x \leftarrow S$ denotes the random process of selecting element x according to distri-

bution D . Moreover, the notation $y \leftarrow A(x)$, where A is an algorithm, denotes the random process of obtaining y when running algorithm A on input x , where the probability space is given by the random coins (if any) of algorithm A . A random variable V will be denoted by $\{R_1; \dots; R_n : v\}$, where v denotes the values that V can assume, and R_1, \dots, R_n is a sequence of random processes generating value v . By $\Pr[R_1; \dots; R_n : E]$ we denote the probability of event E , after the execution of random processes R_1, \dots, R_n . We say that a function in n is *negligible* if it is $\leq n^{-c}$, for all constants c and all sufficiently large n .

2.2 SOME DEFINITIONS

Definition 2.1 (Non-interactive bit-commitment) *Let a be a constant, n be an integer and σ be a public random string of length n^a ; Let (A, B) be a sender-receiver pair. We say that (A, B) is a computationally-secure bit-commitment scheme (resp. perfectly-secure bit-commitment scheme) in the public-random-string model if the following conditions hold:*

1. *Meaningfulness.* For all constants c , each $b \in \{0, 1\}$, and all sufficiently large n ,

$$\Pr[\sigma \leftarrow \{0, 1\}^{n^a}; (com, dec) \leftarrow A(\sigma, b); d \leftarrow B(\sigma, com, dec) : d = b] \geq 1 - n^{-c}.$$

2. *Security.* The families of random variables A_0 and A_1 are computationally (resp. perfectly) indistinguishable, where $A_b = \{\sigma \leftarrow \{0, 1\}^{n^a}; (com, dec) \leftarrow A(\sigma, b) : (\sigma, com)\}$, for $b = 0, 1$.
3. *Binding.* For all algorithms (resp. probabilistic polynomial time algorithms) A' , all constants c , and all sufficiently large n .

$$\Pr[\sigma \leftarrow \{0, 1\}^{n^a}; (com, dec_0, dec_1) \leftarrow A'(\sigma) :$$

$$B(\sigma, com, dec_0) = 0 \wedge B(\sigma, com, dec_1) = 1] < n^{-c}.$$

We remark that the above definition naturally extends to a definition of string commitment scheme. For any string $s = s_1 \circ \dots \circ s_n$, where $s_i \in \{0, 1\}$, the scheme obtained by independently committing to each bit s_i using a secure bit-commitment scheme is a secure string commitment scheme.

Definition 2.2 (Non-interactive equivocable bit commitment) *Let a be a constant, n be an integer and σ be a public random string of length n^a ; Let (A, B) be a bit-commitment scheme in the public random string model. We say that (A, B) is a non-interactive computationally (resp., perfectly) equivocable bit-commitment scheme in the public random string model if there exists an efficient probabilistic algorithm M which, on input 1^n , outputs a 4-tuple $(\sigma', com', dec_0, dec_1)$, satisfying the following:*

1. For $c = 0, 1$, it holds that $B(\sigma', com', dec_c) = c$.
2. For $b = 0, 1$, the families of random variables $A_0 = \{\sigma \leftarrow \{0, 1\}^{n^a}; (com, dec) \leftarrow A(\sigma, b) : (\sigma, com, dec)\}$ and $A_1 = \{(\sigma', com', dec_0, dec_1) \leftarrow M(1^n) : (\sigma', com', dec_b)\}$ are computationally (resp., perfectly) indistinguishable.

As for ordinary commitment, we remark that the above definition naturally extends to a definition of equivocable string commitment scheme, and that for any string $s = s_1 \circ \dots \circ s_n$, where $s_i \in \{0, 1\}$, the scheme is obtained by independently committing to each bit s_i using an equivocable bit commitment scheme.

Let k be an integer and let D be an efficiently sampleable distribution over the set of k -bit strings (represented by its generator). Let R be a relation approximator, that is, an efficient probabilistic algorithm that, given two strings, returns a binary output (algorithm R is supposed to measure the correlation between the two input string). Also, given a committer algorithm, we say that A' is an adversary simulator if, on input D , it outputs a string in $\{0, 1\}^k$ (algorithm A' is supposed to simulate the behavior of an adversary who is not given a commitment as input). Now, consider two experiments: an *a-posteriori* experiment, and an *a-prior* one. In the *a-posteriori* experiment, given a commitment com_1 to a string s_1 , an efficient non-uniform adversary A tries to compute a commitment $com_2 \neq com_1$ which, later, when he is given the decommitment of com_1 , can be decommitted as a string s_2 , have some correlation with string s_1 . In the *a-prior* experiment, an adversary simulator A' commits to a string s_2 , given only the knowledge of D . We consider a non-malleable commitment scheme as a commitment scheme in which for any relation approximator R and for any adversary A , there exists an adversary simulator A' which succeeds "almost as well" as A in returning strings which make R evaluate to 1.

Definition 2.3 (Non-interactive non-malleable string commitment) *Let a be a constant, let (A, B) be a non-interactive string commitment scheme in the public random string model. We say that (A, B) is a non-interactive non-malleable string commitment scheme in the public string model if for every efficient non-uniform algorithm A , there exists an efficient non-uniform adversary simulator A' , such that for all relation approximators R , for all efficiently sampleable distributions D , for all constants c and all sufficiently large n , it holds that $p(A, R) - p'(A', R) \leq n^{-c}$, where the probabilities $p(A, R)$ and $p'(A', R)$ are defined as*

$$\begin{aligned} p(A, R) &= \Pr[\sigma \leftarrow \{0, 1\}^{n^a}; s \leftarrow D; (com_1; dec_1) \leftarrow A(\sigma, s); \\ &\quad com_2 \leftarrow A(\sigma, com_1); dec_2 \leftarrow A(\sigma, com_1, com_2, dec_1) : \\ &\quad B(\sigma, com_1, dec_1) = s \wedge B(\sigma, com_2, dec_2) = t \wedge \\ &\quad com_2 \neq com_1 \wedge R(s, t) = 1]. \\ p'(A', R) &= \Pr[s \leftarrow D; t \leftarrow A'(D) : R(s, t) = 1]. \end{aligned}$$

Notice that the definition of non-interactive non-malleable bit commitment can easily derived from the above.

2.3 SOME LEMMAS

The two following lemmas come from [7].

Lemma 2.1 *In the public random string model, given a non-interactive commitment scheme it is possible to construct a non-interactive equivocable commitment scheme.*

Notice that it is enough to prove the above theorem for the case of single bit-commitment, since already remarked, this would extend to strings using simple independent repetition. We recall the bit-commitment scheme in [8], and the implementation of this scheme in the public random string model is equivocable (its proof is seen in [4, 7]).

bit-commitment scheme 1:

Let $n > 0$ be an integer, and $G : \{0, 1\}^n \rightarrow \{0, 1\}^{3n}$ be a pseudo-random generator agreed upon by the committer A and the receiver B . *The commitment phase* : First B sends a $3n$ -bit uniformly chosen string $R = r_1 \circ \dots \circ r_{3n}$, where each $r_i \in \{0, 1\}$. Then A uniformly chooses an n -bit seed s and computes $G(s) = t_1 \circ \dots \circ t_{3n}$ where each $t_i \in \{0, 1\}$. Then, in order to commit to bit b , for $i = 1, \dots, 3n$, the committer computes bit $c_i = t_i$ if $r_i = 0$ or bit $c_i = t_i \oplus b$ if $r_i = 1$. The commitment key is then string $com = c_1 \circ \dots \circ c_{3n}$, and the decommitment key is $dec = s$. Then A sends the commitment key to B . *The Decommitment phase*. A sends the decommitment key to B . The receiver B , given R , com and s , performs the following test: If $com = G(s)$, B outputs 0; if $com = G(s) \oplus R$, B outputs 1; otherwise, B outputs \perp .

Lemma 2.2 *In the common random string model, for any computationally secure non-interactive commitment scheme, it is possible to construct a computationally secure non-interactive non-malleable commitment scheme.*

We quote the bit-commitment scheme in [7] which has been proven to be a computationally secure non-interactive non-malleable commitment scheme.

bit-commitment scheme 2:

Input to Alice and Bob:

- A security parameter 1^n ;
- an n^a -bit reference string σ , for some constant a ;
- a non-interactive bit-commitment scheme (A, B) ;
- a non-interactive equivocable bit-commitment scheme (C, D) ;
- a pseudo-random generator G ;

Input to Alice: A bit b ;

Instructions for Alice:

A.1 (Commitment to a seed for an authentication key.)

Write σ as $\sigma = \alpha_1 \circ \dots \circ \alpha_n \circ \beta$;

uniformly choose a seed $s \in \{0, 1\}^n$;

let $s_1 \circ \dots \circ s_n$ be its binary expansion;

for $i = 1, \dots, n$,

run algorithm A on input (α_i, s_i) ,

and let $(A-com_i, A-dec_i)$ be its output;

set $A-com = A-com_1 \circ \dots \circ A-com_n$ and let $d_1 \circ \dots \circ d_m$ be

its binary expansion;

write β as $\beta = \beta_{1,0} \circ \beta_{1,1} \circ \dots \circ \beta_{m,0} \circ \beta_{m,1}$.

A.2 (Bit commitment and commitment authentication.)

For $j = 1, \dots, m$.

run algorithm C on input (β_{j,d_j}, b) ,
 and let $(C-com_j, C-dec_j)$ be its output;
 set $C-com = C-com_1 \circ \dots \circ C-com_m$;
 set $q = 2^{|C-com|}$ and $z = G(s)$;
 write z as $z = a \circ c$, where $a, c \in GF(q)$;
 compute $tag = a \cdot (C-com) + c$ (over $GF(q)$).

A.3 (Output.)

Let $Alice-com = (A-com, C-com, tag)$;
 set $A-dec = A-dec_0 \circ \dots \circ A-dec_n$;
 and $C-dec = C-dec_0 \circ \dots \circ C-dec_m$;
 set $Alice-dec = (A-dec, C-dec)$;

Input to Bob:

$Alice-com = (A-com, C-com, tag)$, $Alice-dec = (A-dec, C-dec)$;

Instructions for Bob:

B.1 (Verify the correctness of the decommitment.)

For $i = 1, \dots, n$,
 verify that $B(\alpha_i, A-com_i, A-dec_i) \neq \perp$;
 for $i = 1, \dots, n$,
 let $s_i = B(\alpha_i, A-com_i, A-dec_i)$, and $s = s_1 \circ \dots \circ s_n$.
 let $d_1 \circ \dots \circ d_m$ be the binary expansion of $A-com$.
 verify that there exists $b \in \{0, 1\}$ such that
 $D(\beta_{j,d_j}, C-com_j, C-dec_j) = b$, for $j = 1, \dots, m$.
 set $q = 2^{|C-com|}$ and $z = G(s)$;
 write z as $z = a \circ c$, where $a, c \in GF(q)$;
 verify that $tag = a \cdot (C-com) + c$ (over $GF(q)$).

B.2 (output.)

If any verification is not satisfied that output \perp and halt
 else output the bit b .

3. NON-INTERACTIVE EQUIVOCAL NON-MALLEABLE COMMITMENT

In this section we will prove our main results.

If the non-interactive bit-commitment (A, B) in bit commitment scheme 2 is a non-interactive equivocable bit-commitment, then the following theorem holds.

Theorem 3.1 *If the non-interactive bit-commitment (A, B) in bit commitment scheme 2 is a non-interactive equivocable bit-commitment, then the bit-commitment scheme 2 is the non-interactive equivocable non-malleable bit-commitment scheme.*

Sketch of Proof: We need to show an efficient simulator M , which on input 1^n , generates a 4-tuple $(\sigma', com', dec_0, dec_1)$ satisfying properties 1 and 2 of Definition 2.2.

The Algorithm M . If the bit-commitment scheme (A, B) in bit-commitment scheme 2 is the bit-commitment scheme in [8], then it is possible to construct a non-interactive equivocable bit-commitment scheme by Lemma 2.1, that is, there exists an efficient simulator M_1 which generates a 4-tuple $(\sigma'_1, com'_1, dec_{10}, dec_{11})$ satisfying properties 1 and 2 of Definition 2.2. Furthermore, the bit-commitment scheme (C, D) in bit-commitment scheme 2 is a non-interactive equivocable bit-commitment scheme, so the following holds: there exists an efficient simulator M_2 which generates a 4-tuple $(\sigma'_2, com'_2, dec_{20}, dec_{21})$ satisfying properties of Definition 2.2.

Let $M = (M_1, M_2)$, that is, the simulator M_1 and simulator M_2 consist sequentially the simulator M . Set $\sigma' = (\sigma'_1, \sigma'_2)$, $com' = (com'_1, com'_2)$, $dec_0 = (dec_{10}, dec_{20})$ and $dec_1 = (dec_{11}, dec_{21})$. Hence we obtain a simulator M which generates the 4-tuple $(\sigma', com', dec_0, dec_1)$. In the following, we will prove the simulator M to satisfy properties 1 and 2 of Definition 2.2.

M can open both as 0 and as 1. Clearly, M_1 can open both as 0 and as 1, and M_2 do so. In construction of simulator M , M_1 and M_2 are independent, so M can open both as 0 and as 1. (In fact, the bit-commitment scheme (A, B) only commits to a seed for an authentication key, and it is unrelated to the committed bit, hence, the last commitments of bit is only controlled by bit-commitment scheme (C, D)).

M 's output is indistinguishable from a real execution. Let us recall the definition of the two random variables in Definition 2.2: $Alice_0 = \{\sigma \leftarrow \{0, 1\}^{n^a}, (com, dec) \leftarrow Alice(\sigma, b) : (\sigma, com, dec)\}$, and $Alice_1 = \{(\sigma', com', dec_0, dec_1) \leftarrow M(1^n) : (\sigma', com', dec_b)\}$ and $b \in \{0, 1\}$. Assume, for the sake of contradiction, that there exists a probabilistic polynomial time algorithm D , which distinguishes $Alice_0$ from $Alice_1$, with probability at least n^{-c} , for some constant c and infinitely many n . Because the simulator M is composed of M_1 and M_2 , one of the following two case must hold:

1) for simulator M_1 , there exists a probabilistic polynomial time algorithm D_1 , which distinguishes A_0 from A_1 , with probability of least n^{-c} , for some constant c and infinitely many n . where $A_0 = \{\sigma_1 \leftarrow \{0, 1\}^{n^a}, (com_1, dec_1) \leftarrow A(\sigma_1, b) : (\sigma_1, com_1, dec_1)\}$, $A_1 = \{(\sigma'_1, com'_1, dec_{10}, dec_{11}) \leftarrow M_1(1^n) : (\sigma'_1, com'_1, dec_{1b})\}$. and $b \in \{0, 1\}$.

2) for simulator M_2 , there exists a probabilistic polynomial time algorithm D_2 , which distinguishes C_0 from C_1 , with probability of least n^{-c} , for some constant c and infinitely many n , where $C_0 = \{\sigma_2 \leftarrow \{0, 1\}^{n^a}, (com_2, dec_2) \leftarrow A(\sigma_2, b) : (\sigma_2, com_2, dec_2)\}$, $C_1 = \{(\sigma'_2, com'_2, dec_{20}, dec_{21}) \leftarrow M_2(1^n) : (\sigma'_2, com'_2, dec_{2b})\}$. and $b \in \{0, 1\}$.

Now, we will prove that the case 1) generates a contradiction, and the proof of the case 2) is similar.

We show the existence of a probabilistic polynomial time algorithm E , which, using D_1 as a subroutine, is able to distinguish the output of pseudo-random generator G from a totally random string with probability at least n^{-c} , for some constant c and infinitely many

n . Algorithm E works as follows: on input a string y , it randomly chooses a seed $s \in \{0, 1\}^n$ and sets $u = G(s)$ and $R = u \oplus y$. Now, it randomly chooses $v \leftarrow \{y, u\}$, and runs algorithm D_1 on input (R, v, s) . Algorithm D_1 returns a bit c , denoting that it guesses that the triple (R, v, s) is distributed according to A_c . Finally, algorithm E outputs 'pseudo-random' if $c = 1$ and 'random' if $c = 0$. By observing that the triple (R, v, s) is distributed as A_0 if y is totally random or as A_1 if y is output by G , we derive that the probability that algorithm E distinguishes whether y is random or pseudo-random is the same as the probability that algorithm D distinguishes A_0 from A_1 . **end**

Clearly, the results in Lemma 2.1, Lemma 2.2 and Theorem 3.1 are enough to prove our main result in the following Theorem.

Theorem 3.2 *In the public random string model, for any computationally secure non-interactive commitment scheme, it is possible to construct a computationally secure non-interactive equivocable non-malleable commitment scheme.*

Remarks: As already remarked, given a secure bit-commitment scheme, it is possible to obtain a string commitment scheme by repeating independent executions of the original bit-commitment scheme. It should be observed that this transformation does not preserve the non-malleability property. Let us try to get convinced that this is indeed the case. First, let (A, B) be a bit-commitment scheme, consider a string s of two bits s_0, s_1 , and a bit-commitment scheme (C, D) constructed as a double repetition of (A, B) , each having as input a different bit of s . We see that even if (A, B) is non-malleable then (C, D) is malleable. Specifically, consider the algorithm C' that, after seeing the commitment com_0, com_1 to s_0, s_1 by C , outputs com_1, com_0 , namely, he just swaps the two single bit-commitment keys. Clearly, string s_1, s_0 is 'related' to the original string s , and therefore algorithm C' shows that scheme (C, D) is not non-malleable. Notice that our reasoning does not depend on whether we are in the interactive or non-interactive setting. In [7], the bit-commitment scheme 2 is properly modified in order to preserve non-malleability with independent repetitions. and we omit it.

By Theorem 3.2, we know how to construct a non-interactive equivocable non-malleable commitment scheme, if bit-commitment scheme (A, B) in bit-commitment scheme 2 is non-interactive and equivocable. In cryptographic fields, the equivocable commitment scheme has vast applications, for example, zero-knowledge proof in an asynchronous setting [4]. Our commitment scheme has not only equivocability but also non-malleability, so its applications must be bigger. In the following section, we will consider its application.

4. CONCURRENT ZERO-KNOWLEDGE

The notion of zero-knowledge proof systems was introduced by Goldwasser, Micali and Rackoff [9]. Since their introduction, zero-knowledge proofs have been proven to be very useful as a building block in the construction of cryptographic protocols, especially after Goldreich, Micali and Wigderson [10] have shown that all languages in NP admit zero-knowledge proofs. Due to their importance, the efficiency of zero-knowledge protocols has received considerable attention. However, recently, a lot of attention has been paid to the setting where many concurrent executions of the same protocol take place (say, on the Internet).

The general case concurrent zero-knowledge in an asynchronous setting was considered by *Dwork, Naor and Sahai* in [11]. In [4], *Di Crescenzo and Ostrovsky* considered a distributed model with preprocessing phase and a proof phase. In this model, based on complexity assumptions, they constructed the concurrent zero-knowledge for all languages in NP .

In this section, we recall the notion of zero-knowledge and concurrent zero-knowledge; use the commitment scheme in Theorem 3.2 to implement concurrent zero-knowledge; analyze our model's properties; at last, compare with concurrent zero-knowledge of [4].

Zero-knowledge proofs systems: A zero-knowledge proof system for a language L is an interactive system for L in which, for any $x \in L$, and any possibly malicious probabilistic polynomial-time verifier V' , no information is revealed to V' that he could not compute alone before running the protocol. This is formalized by requiring, for each V' , the existence of an efficient simulator $S_{v'}$ which outputs a transcript 'indistinguishable' from the view of V' in the protocol. There exist three notions of zero-knowledge: computational, statistical and perfect.

Concurrent zero-knowledge proof systems with preprocessing. Informally speaking, the concurrent model describes a distributed model in which several parties can run concurrent executions of some protocols. In real-life distributed system, the communication is not necessarily synchronized; more generally, the model makes the worst-case assumption that the communication in the system happens in an asynchronous manner; this means that there is no fixed bound on the amount of time that a message takes to arrive to its destination. This implies that, for instances, the order in which messages are received during the execution of many concurrent protocols can be arbitrarily different from the order in which they were sent. Such variation in the communication model poses a crucial complication into designing a zero-knowledge protocol, since the possibly arbitrary interleaving of messages can help some adversary to break the zero-knowledge property. In fact, as a worst case assumption, one may assume that the ordering in which messages are received can be decided by the adversary.

We consider a distributed model, with two distinguished sets of parties: a set $P = \{P_1, \dots, P_q\}$ of provers and a set $V = \{V_1, \dots, V_q\}$ of verifiers, where P_i is connected to V_i , for $i = 1, \dots, q$. Let (A, B) be a zero-knowledge proof system. At any time a verifier V_i may decide to run protocol (A, B) ; therefore, for any fixed time, there may be several pairs of prover and verifier running (A, B) . The adversary A is allowed to corrupt all the verifiers. Then A can be formally described as a probabilistic polynomial-time algorithm that, given the history so far, returns an index i and a message m_i , with the meaning that the (corrupted) verifier V_i , for $i \in \{1, \dots, q\}$, is sending message m_i to prover P_i . We assume wlog that P_i is required to send his next message m_i and before receiving a new message from A . We now define the view of the adversary A . First we define a q -concurrent execution of (A, B) as the possibly concurrent execution of q instances of protocol (A, B) , where all verifiers are controlled by A . Also, we define the q -concurrent transcript of a q -concurrent execution of (A, B) as a sequence containing the random tapes of verifiers V_1, \dots, V_q and all messages appearing on the communication tapes of P_1, \dots, P_q and V_1, \dots, V_q , where the ordering of such messages is determined by the adversary corrupting all the verifiers. The notation $(T; y_{11}, y_{12}, \dots, y_{1q}, y_{2q}) \leftarrow ((P_1(p_1), V_1(v_1))(x_1), \dots, (P_q(p_q), V_q(v_q))(x_q))$ denotes the random process of running a q -concurrent execution of interactive protocol (A, B) , where each P_i has

private input p_i , each V_i has private input v_i , x_i is P_i and V_i 's common input, y_{1i} is P_i 's output, y_{2i} is V_i 's output, and T is the q -concurrent transcript of such execution (we assume *wlog* that the output of both parties P_i and V_i at the end of an execution of the interactive protocol (A, B) contains a transcript of the communication exchanged between P_i and V_i during such execution). Then the view of A , denoted as $view_A(x)$ is the transcript T output by the random process $(T; y_{11}, y_{12}, \dots, y_{1q}, y_{2q}) \leftarrow ((P_1(p_1), A(v_1))(x_1), \dots, (P_q(p_q), A(v_q))(x_q))$, where $x = (x_1, \dots, x_q)$. Finally, a proof system (A, B) for language L is *concurrent zero-knowledge* if for any probabilistic polynomial time algorithm A , there exists an efficient algorithm S_A such that for any polynomial $q(\cdot)$, and any x_1, \dots, x_q , where $q = q(n)$, and $|x_1| = \dots = |x_q| = n$, the two distributions $S_A(x)$ and $View_A(x)$ are distinguishable.

We use the definition of *Concurrent zero-knowledge with preprocessing* in [4]. In their definition, a concurrent zero-knowledge of an interactive protocol with preprocessing $(A1, B1)(A2, B2)$ is divided into two phases: a preprocessing phase and a proof phase. In the preprocessing phase there is a concurrent execution of the preprocessing pair $(A1, B1)$ and in the proof phase there is a concurrent execution of the proof pair $(A2, B2)$. The requirements are, as before, completeness, soundness and concurrent zero-knowledge. Now we give their formal definition of concurrent zero-knowledge proof systems with preprocessing.

Definition 4.1 Concurrent zero-knowledge proof systems with preprocessing: Let $(A, B) = (A1, B1)(A2, B2)$ be an interactive protocol with preprocessing. We say that (A, B) is a concurrent (computational) (statistical) (perfect) zero-knowledge proof systems with preprocessing for language L if the following holds:

1. *Completeness:* for any $x \in L$, it holds that $Prob[(\alpha, \beta) \leftarrow (A1, B1)(1^{|x|}); (t, (t, out)) \leftarrow (A2(\alpha), B2(\beta))(x) : out = ACCEPT] \geq 1 - 2^{-|x|}$.
2. *Soundness:* For any $x \notin L$, and any $(A1', A2')$, it holds that $Prob[(\alpha, \beta) \leftarrow (A1', B1)(1^{|x|}); (t, (t, out)) \leftarrow (A2'(\alpha), B2(\beta))(x) : out = ACCEPT] < 2^{-|x|}$.

3. *Concurrent Zero-knowledge:* For each probabilistic polynomial time algorithm $A = (A_1, A_2)$, there exists an expected polynomial time simulator algorithm S_A such that for any polynomial $q(\cdot)$, for each $x_1, \dots, x_q \in L$, where $|x_1| = \dots = |x_q| = n$ and $q = q(n)$, the two distribution $S_A(x)$ and $View_A(x)$ are (computationally)(statistically)(perfectly) indistinguishable, where $View_A(x) = \{(T_1, \alpha_1, \beta_1, \dots, \alpha_q, \beta_q) \leftarrow ((P_1, A_1), \dots, (P_q, A_1))(1^n); (T_2, \cdot, \cdot, \dots, \cdot, \cdot) \leftarrow ((P_1(\alpha_1), A_2(\beta_1))(x_1), \dots, (P_q(\alpha_q), A_2(\beta_q))(x_q)) : (T_1, T_2, \beta_1, \dots, \beta_q)\}$

denotes the view of A on input $x = (x_1, \dots, x_q)$, and where P_1, \dots, P_q run algorithms $A1$ in the preprocessing phase and $A2$ in the proof phase.

In the following part, we consider the properly modified Proof System $((P1, V1), (P2, V2))$ (abbreviated *PS*) in [4], which has been proven to preserve concurrent zero-knowledge with preprocessing.

The Proof System $((P1, V1), (P2, V2))$

Input to P1 and V1: 1^n , where n is a positive integer.

Instructions for P1 and V1 (preprocessing phase):

1. $V1$ uniformly chooses $u_1, \dots, u_m \in \{0, 1\}^{3n}$ and commits to them using scheme (C, D) (possibly interacting with $P1$);
2. $P1$ uniformly chooses $v_1, \dots, v_m \in \{0, 1\}^{3n}$ and sends them to $V1$;
3. $V1$ decommits u_1, \dots, u_m using scheme (C, D) (possibly interacting with $P1$);
4. If u_1, \dots, u_m are not properly decommitted, $P1$ halts;
5. $P1$ sets α equal to the transcript so far and outputs: α ;
6. $V1$ sets β equal to the transcript so far and outputs: β .

Input to $P2$: x, α , where $|x| = n$.

Input to $V2$: x, β , where $|x| = n$.

Instructions for $P2$ and $V2$ (proof phase):

1. $P2$ sets $mes = A(x)$ and uniformly chooses $d_1, \dots, d_m \in \{0, 1\}$; Then commits to them using a equivocable commitment scheme (S, R) , and obtains commitments $com_1, com_2, \dots, com_m$; At last, sends $mes, com_1, com_2, \dots, com_m$ to B .
2. $V2$ uniformly chooses $b \in \{0, 1\}^m$ and sends b to $P2$;
3. $P2$ decommits d_1, \dots, d_m by sending s_1, \dots, s_m to $V2$; then sets $d = d_1 \circ \dots \circ d_m, c = b \oplus d, ans = A(x, mes, c)$ and sends ans to $V2$;
4. $V2$ verify the message. If $B(x, mes, c, ans) = \text{ACCEPT}$ then $V2$ outputs: ACCEPT else $V2$ outputs: REJECT .

In [4], *Di Crescenzo* and *Ostrovsky* have proven that above PS preserves zero-knowledge in the asynchronous setting. Usually, because of the presence of potentially bad interleaving among the polynomially many concurrent executions, proving a protocol to be concurrent zero-knowledge becomes a problem. In particular, such interleavings may ask the simulator for too many(eventually, more than polynomial) rewindings in order to be able to succeed in simulating the protocol. Therefore, *Di Crescenzo* and *Ostrovsky* used a technique that allowed to simulate part of protocol PS without performing any rewinding part of protocol PS used a special zero-knowledge property. For example, in PS , they separated the preprocessing phases of all protocols from the proof phases of all protocols and obtain concurrent zero-knowledge. This is achieved by using the tool of equivocable commitment scheme (S, R) . However, the equivocable commitment scheme used by them is provably malleable(informally, in the context of commitment the additional requirements is that given the commitment it is possible to generate a different commitment so that the respective *bit* committed are related) so that there exist some chances which the attacker can break zero-knowledge.

in fact, there exists some commitment schemes are malleable. for example, the bit-commitment scheme 1 is based on pseudo-random generators. Now we show that it is malleable.

Given a random string R and a commitment com to a bit b , an attacker can commit to bit $1 - b$ by sending the commitment $com' = com \oplus R$. The decommitment key dec opening commitment key com as b also opens commitment key com' as $1 - b$.

In order to avoid the case, we must look for more robust commitment scheme than (S, R) in PS . Clearly, The commitment scheme in Theorem 3.2 is a good refill, because it is non-interactive equivocable non-malleable. Our protocol PS , which uses the non-interactive equivocable non-malleable commitment scheme, has the following properties:

1. the PS preserves the concurrent zero-knowledge. Because the equivocability of bit-commitment holds.
2. the attacker can not break the commitment scheme, and can not break the zero-knowledge, because the non-malleability of bit-commitment holds.
3. the construction of our protocol is more simpler than that of the protocol PS , because our commitment scheme is non-interactive.

References

- [1] D. Dolev, C. Dwork, and M. Naor, Non-malleable Cryptography, *Proc. of STOC 91*.
- [2] M. Blum, P. Feldman, and S.Micali, Non-interactive Zero-knowledge and its Applications, *Proc. of STOC 88*.
- [3] M. Blum, A. De Santis, S. Micali, and G. Persiano, Non-interactive Zero-knowledge, *SIAM Journal of computing*, vol. 20, no.6, Dec.1991, pp. 1084-1118
- [4] G. Di Crescenzo, and R. Ostrovsky, On Concurrent Zero-knowledge with Pre-processing.
- [5] A. De Santis, and G. Persiano, Zero-knowledge Proofs of Knowledge Without Interaction. *Proc. of FOCS 92*.
- [6] M. Bellare, and P. Rogaway, Random Oracles are Practical: A paradigm for Designing Efficient Protocols. *Proc. of ACM Conference on Computer and Communication Security*,1993.
- [7] G. Di Crescenzo, Y. Ishai, and R. Ostrovsky, Non-interactive and Non-Malleable Commitment.
- [8] M. Naor, Bit Commitment using Pseudo-Randomness, *Proc. of CRYPTO 89*.
- [9] S. Goldwasser, S. Micali, and C. Rackoff, The Knowledge Complexity of Interactive Proof-Systems, *SIAM Journal on Computing*, vol.18, n. 1, February 1989.
- [10] O. Goldreich, S. Micali, and A.Wigderson, Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proofs Systems, *Journal of the ACM*, vol.38, n.1, 1991, pp.691-729.
- [11] D.Dolev, C. Dwork, and M. Naor, Concurrent Zero-Knowledge, *Proc. of STOC 98*.