Obtaining Efficient Fully Simulatable Oblivious Transfer from General Assumptions

Bernardo M. David[1], **Anderson C. A. Nascimento**[1], **Rafael Tonicelli**[1]

[1]Department of Electrical Engineering, University of Brasilia.
Campus Universitario Darcy Ribeiro,Brasilia, CEP: 70910-900, Brazil

bernardo.david@redes.unb.br, andclay@ene.unb.br, tonicelli@redes.unb.br

Abstract. *We introduce a general construction of fully simulatable oblivious transfer based on lossy encryption. Furthermore, we extend the common definition of lossy encryption by introducing the notion of computationally lossy encryption. If the cryptosystem used is computationally lossy, our general construction yields oblivious transfer protocols with computational security for both parties. Otherwise, when regular statistically lossy cryptosystems are employed in this construction, it yields oblivious transfer protocols with statistical security for the sender. The construction introduced in this paper is realizable from re-randomizable, homomorphic and lossy cryptosystems in general. Thus, it yields specific constructions based on different assumptions, such as DDH, LWE and McEliece. Moreover, it proves the equivalence of fully simulatable oblivious transfer and lossy encryption.*

1. Introduction

Oblivious transfer (OT), a cryptographic primitive introduced by Rabin [Rabin 1981], is of great importance in the design of secure two-party and multiparty computation protocols.There exist many variants of OT, each one suitable for a given kind of application. In the present work, we concentrate ourselves on a variant called *one-out-of-two oblivious transfer*, denoted by $\binom{2}{1}$-OT. In this variant, a sender (Alice) inputs two bits $b_0,\ b_1$ and a receiver (Bob) inputs a choice bit σ. At the end of the protocol, Alice receives nothing and Bob receives the bit b_σ. Loosely speaking, an OT protocol is said to be private if the sender learns no information on the receiver's choice σ, while the receiver gets information concerning at most one of the sender's inputs.

It has been proven that oblivious transfer enjoys a property called *completeness*, meaning that any function can be securely computed if the parties are given black-box access to OT [Kilian 1988]. Since OT serves as a building block for a wide variety of secure protocols, it is desirable to have OT protocols that achieve a strong notion of security against an unrestricted adversarial model. Regarding the adopted notion of security, it is of particular interest to design OT protocols that are *fully-simulatable*, that is, secure in the *real/ideal model simulation paradigm*. It is a well-known fact that OT protocols proven secure in the simulation-based paradigm are secure under sequential composition and, consequently, can truly be used as building blocks in more complex protocols. Regarding the adopted adversarial model, it is desirable for an OT protocol to be resistant against a *malicious adversary*. In contrast to a semi-honest adversary (who follows the protocol, but may try to acquire more information than it is allowed to know), a malicious

adversary may arbitrarily deviate from the protocol specifications and, thus, represents a more powerful adversarial model.

In spite of being a fundamental cornerstone in two- and multi-party computation, there are only a few results of efficient OT constructions that are secure against malicious adversaries under the *real/ideal model simulation paradigm* without random oracles [Lindell 2008, Green and Hohenberger 2007, Camenisch et al. 2007]. In [Camenisch et al. 2007] the authors introduce constructions based on q-power DDH and q-strong Diffie-Hellman, which are strong and relatively non-standard assumptions. In [Green and Hohenberger 2007], a construction based on the Decisional Bilinear Diffie-Hellman assumption is introduced. The first construction based on standard assumptions is given in [Lindell 2008], which introduces protocols based on the DDH assumptions, smooth projective hashing and general homomorphic cryptosystems (which is an assumption much stronger than lossy encryption, being realizable under much more limited number of underlying computational assumptions).

1.1. Contributions

In this paper, we present the following significant contributions:

- An efficient fully-simulatable oblivious transfer protocol based on a general assumption: *Lossy Encryption* [Hemenway et al. 2009, Bellare et al. 2009]. In this construction we utilize techniques from the DDH based efficient fully simulatable protocol presented in [Lindell 2008]. It is realizable from a broad number of general assumptions (such as smooth projective hashing and re-randomization), computational assumptions (such as DDH, LWE) and also the McEliece assumptions under the extra assumption of the existence of perfectly binding and perfectly hiding commitments.
- Our construction proves the equivalence between fully simulatable oblivious transfer and several flavors of lossy encryption, since the converse is shown in [Hemenway et al. 2009].
- We introduce computationally lossy encryption, which is realizable under a broader number of assumptions than statistically lossy encryption, and show that the proposed general construction achieves computational security for both parties in the case that such a cryptosystem is employed.
- We unify all current constructions of efficient fully simulatable oblivious transfer in the plain model, since lossy encryption (computational or statistical) is realizable under all of the assumptions previously used to construct fully simulatable oblivious transfer in the plain model, such as: smooth projective hashing re-randomizable cryptosystems, homomorphic cryptosystems, DDH, q-power DDH, q-strong Diffie-Hellman and Bilinear Diffie-Hellman.

In summary, the main contribution of this paper is to provide a general construction for fully-simulatable oblivious transfer based on the sole assumption of lossy encryption and a property enjoyed by many constructions of such cryptosystems. This construction is realizable under several well known computation assumptions, including factorization, discrete logarithm, lattice and coding theory problems. Hence, it unifies all current constructions of efficient fully simulatable oblivious transfer in the plain model.

2. Preliminaries

Hereupon, we will denote by $x \in_R D$ an uniformly random choice of element x over its domain D; by $\oplus$ a bit-wise exclusive OR of strings; and by $a \mid b$ the concatenation of string a with string b. All logarithms are to the base 2. For a ***PPT*** machine A, we use $a \xleftarrow{\$} A$ to denote running the machine A and obtaining an output, where a is distributed according to the internal randomness of A.

If X and Y are families of distributions indexed by a security parameter λ, we use $X \overset{s}{\approx} Y$ to mean the distributions X and Y are *statistically close*, *i.e.*, for all polynomials p and sufficiently large λ, we have $\sum_x |Pr[X = x] - Pr[Y = x]| < 1$. Two sequences $X_n, n \in \mathbb{N}$ and $Y_n, n \in \mathbb{N}$ of random variables are said to be *computationally indistinguishable*, denoted by $X \overset{c}{=} Y$, if for every non-uniform probabilistic polynomial-time distinguisher D there exists a negligible function $\epsilon(\cdot)$ such that for every $n \in \mathbb{N}$, $\mid Pr[D(X_n) = 1] - Pr[D(Y_n) = 1] \mid < \epsilon(n)$.

2.1. Real/Ideal Model Simulation Paradigm

The *real/ideal model paradigm* has been extensively used to analyse the security of protocols under sequential composition. In this model, security is analysed by comparing real protocol execution with an ideal execution. In the ideal execution, the parties send their private inputs to a trusted party that computes the desired functionality through confidential and authenticated channels. After receiving the inputs, the trusted party computes the function and returns the output assigned to each party. In the real execution, the parties interact directly through the protocol. Intuitively, if all attacks feasible in the real model are also feasible in the ideal model, the protocol is considered secure.

Ideal Model Execution. An ideal $\binom{2}{1}$-OT functionality is formally defined as function f with two inputs and one output. The sender Alice inputs two bits $(b_0,\ b_1)$, while the receiver Bob inputs a bit σ. After the protocol is run, Alice receives no output (denoted by the empty string λ), and Bob receives b_σ. This is denoted as: $f : \{0,1\}^2 \times \{0,1\} \rightarrow \{0,1\}$, such that $f((b_0,\ b_1), \sigma) = (\lambda,\ m_\sigma)$.

Considering two two parties P_a (Alice) and P_b (Bob) that have access to a trusted third party $\mathcal{T}$, the ideal oblivious transfer functionality is described bellow.

Ideal OT Execution

Input generation. Party P_a is activating upon receiving a pair $(b_0,\ b_1) \in \{0,1\}^2$ and party P_b is activated upon receiving a bit σ.

Transmission of inputs to $\mathcal{T}$. An honest participant sends its unaltered output to the trusted party $\mathcal{T}$. A malicious participant may abort (sending $\perp$ to $\mathcal{T}$) or send any other input to $\mathcal{T}$.

Output computation by $\mathcal{T}$. If the functionality $\mathcal{T}$ receives $\perp$ from any of the parties, then it sends $\perp$ to both parties and halts. Else, upon receiving $(b'_0,\ b'_1)$ from P_a and σ' from P_b, $\mathcal{T}$ sends $b'_{\sigma'}$ to party P_b and halts.

Outputs. An honest party always outputs the message as received from $\mathcal{T}$ ($\perp$ or nothing in the case of P_a, and $\perp$ or $b'_{\sigma'}$ in the case of P_b. A corrupted party can output an arbitrary ***PPT*** function of its initial input and the message obtained from the trusted party.

Let f an ideal oblivious transfer functionality and let $\overline{B} = (B_1,\ B_2)$ denote an admissible pair (*i.e.* at least one of the parties is honest) of non-uniform probabilistic expected polynomial-time machines (representing parties in the ideal model). The joint execution of f under $\overline{B}$ in the ideal model on inputs $((b_0, b_1), \sigma)$, denoted by $\text{IDEAL}_{f,\overline{B}}((b_0, b_1), \sigma)$, is defined as the resulting output pair and protocol transcript obtained by B_1 and B_2 after the ideal execution.

Real Model Execution. for this execution, no trusted party is available and the parties interact directly. A corrupted party may adopt any arbitrary strategy implementable by non-uniform *PPT* machines. Let π denote a two-party protocol and let $\overline{A} = (A_1,\ A_2)$ denote a pair of non-uniform *PPT* machines (representing parties in the real model). The joint execution of π under $\overline{A}$ in the real model on inputs $((b_0, b_1), \sigma)$, denoted by $\text{REAL}_{\pi,\overline{A}}((b_0, b_1), \sigma)$, is defined as the resulting output pair and protocol transcript obtained by A_1 and A_2 after the protocol execution.

Adversarial Model. In this paper, we consider the malicious adversarial model, where a dishonest party may arbitrarily disrupt the protocol execution (for instance, a malicious party is allowed to deviate from the protocol). Additionally, we assume the *static corruption model*, where parties have fixed a behavior throughout protocol execution.

Enlightened by the previous definitions, we can now formalize the notion of securely implementing an OT protocol in the simulation-based paradigm.

Definition 1. *Consider an ideal OT functionality f and a two-party protocol π in the real model. The protocol π is said to securely implement an OT protocol if for every pair of admissible non-uniform PPT machines $\overline{A} = (A_1,\ A_2)$ for the real model, there exists a pair of admissible non-uniform probabilistic expected polynomial-time machines $\overline{B} = (B_1,\ B_2)$ for the ideal model, such that for every $b_0,\ b_1 \in \{0,1\}$ and every $\sigma \in \{0,1\}$,*

$$\left\{\text{IDEAL}_{f,B}(n, (b_0, b_1), \sigma) \stackrel{c}{\equiv} \text{REAL}_{\pi,A}(n, (b_0, b_1), \sigma)\right\}$$

In order to achieve constant-round protocols it is necessary to allow the ideal adversary and simulators to run in expected polynomial time [Barak and Lindell 2004].

3. Lossy Encryption

Lossy encryption [Hemenway et al. 2009, Bellare et al. 2009] expands on the definition of Dual Mode Encryption [Peikert et al. 2008], a type of cryptosystem with two types of public keys, which specify two modes of operation: a *messy* mode and a decryption mode. In the decryption mode, the cryptosystem behaves normally and it is possible to decrypt a message encrypted with a given public key using the corresponding secret key. However, in the messy mode, the encrypted information is statistically lost.

A lossy cryptosystem is defined as a type of cryptosystem with two types of public keys, *injective* and *lossy* keys, which specify different results of encryption. If injective keys are used, the cryptosystem behaves regularly (correctly decrypting ciphertexts with the right secret key) while in the *lossy* mode, the ciphertexts generated by the encryption algorithm are independent from the plaintext messages,causing information to be statistically lost. It is also required that lossy keys are indistinguishable from injective keys by efficient adversaries.

It has been shown that it is possible to obtain lossy cryptosystems from oblivious transfer, re-randomization and smooth projective hashing [Hemenway et al. 2009]. Thus, our construction of fully simulatable oblivious transfer based on lossy encryption proves that oblivious transfer and lossy encryption are equivalent.

We now present a formal definition of Lossy Encryption similar to the definition given in [Hemenway et al. 2009]:

Definition 2. *A lossy public-key encryption scheme is a tuple* $(\mathtt{G}, \mathtt{E}, \mathtt{D})$ *of efficient algorithms such that*

- $\mathtt{G}(1^\lambda, \mathsf{inj})$ *outputs keys* (pk^{inj}, sk^{inj})*, keys generated by* $\mathtt{G}(1^\lambda, \mathsf{inj})$ *are called injective keys.*
- $\mathtt{G}(1^\lambda, \mathsf{lossy})$ *outputs keys* (pk^{lossy}, sk^{lossy})*, keys generated by* $\mathtt{G}(1^\lambda, \mathsf{lossy})$ *are called lossy keys.*

$\mathtt{E}(pk, m)$ *is an encryption algorithm that takes as input a public key and a plain-text message, outputting a ciphertext.*

$\mathtt{D}(sk, c)$ *is a decryption algorithm that takes as input a secret key and ciphertext, outputting a plain-text message.*

Additionally, the algorithms must satisfy the following properties:

- ***Correctness on injective keys****. For all plaintexts* $x \in X$*,*

$$Pr\left[(pk^{inj}, sk^{inj}) \xleftarrow{\$} \mathtt{G}(1^\lambda, \mathsf{inj}); r \xleftarrow{\$} coins(\mathtt{E}) : \mathtt{D}(sk^{inj}, \mathtt{E}(pk^{inj}, x, r)) = x\right] = 1$$

- ***Indistinguishability of keys****. In lossy mode, public keys are computationally indistinguishable from those in the injective mode given no previous information. Specifically, if* $proj : (pk, sk) \to pk$ *is the projection map, then*

$$\left\{proj(\mathtt{G}(1^\lambda, \mathsf{inj}))\right\} \overset{c}{\approx} \left\{proj(\mathtt{G}(1^\lambda, \mathsf{lossy}))\right\}$$

- ***Lossiness of lossy keys****. If* $(pk^{lossy}, sk^{lossy}) \xleftarrow{\$} \mathtt{G}(1^\lambda, \mathsf{lossy})$ *, then for all* $x_0, x_1 \in X$*, the statistical distance between the distributions* $\mathtt{E}(pk^{lossy}, x_0, R)$ *and* $\mathtt{E}(pk^{lossy}, x_1, R)$ *is negligible in* λ*.*
- ***Openability****. If*$(pk^{lossy}, sk^{lossy}) \xleftarrow{\$} \mathtt{G}(1^\lambda, \mathsf{lossy})$*, and* $r \xleftarrow{\$} coins(\mathtt{E})$ *, then for all* $x_0, x_1 \in X$ *with overwhelming probability, there exists* $r' \in coins(\mathtt{E})$ *such that* $\mathtt{E}(pk^{lossy}, x_0, r) = \mathtt{E}(pk^{lossy}, x_1, r')$*. In other words, there is an (unbounded) algorithm* `opener` *that can open a lossy ciphertext to any arbitrary plaintext with all but negligible probability.*

Additionally, we require that there exists an efficient algorithm that distinguishes between lossy and injective public keys given the corresponding secret key. Such algorithm is formally denoted as:

- $\mathtt{KD}(sk, pk)$ is a ***PPT*** algorithm that receives as input a key pair (sk, pk) and outputs 0 if the public key is lossy. Otherwise, it outputs 1.

This property is valid for many flavors of lossy encryption such as the general constructions based on re-randomization and smooth projective hashing [Hemenway et al. 2009].

However, for the sake of brevity, we will give formal proof only for the re-randomization based construction, which is realized by many underlying computational assumptions. The algorithm for the smooth projective hashing based construction follows trivially from the fact that the key generation algorithm outputs an empty secret key if a lossy public key is generated.

3.1. Computationally Lossy Encryption

In the present work, we also consider a variation of common statistically lossy encryption which we call computationally lossy encryption. In a computationally lossy cryptosystem, the distribution of ciphertexts generated under a lossy key is computationally indistiguishable from the uniform distribution of ciphertexts (*i.e.* information is lost only computationally). Such cryptosystems preserve all properties of statistically lossy cryptosystems but the lossiness of key, which in this case is computational:

- **Lossiness of lossy keys**. If $(pk^{lossy}, sk^{lossy}) \xleftarrow{\$} \mathtt{G}(1^{\lambda}, \mathsf{lossy})$, then for all $x_0, x_1 \in X$, the distributions $\mathtt{E}(pk^{lossy}, x_0, R)$ and $\mathtt{E}(pk^{lossy}, x_1, R)$ are computationally indistinguishable in λ.

Computationally lossy encryption is interesting since it yields an OT protocol with computational security for both parties, a fact that has been previously observed only in [Dowsley et al. 2008], which is not secure under sequential composition. Furthermore, such a construction may be realized under a broader number of assumptions than statistically lossy encryption allows. For example, it can be trivially obtained under the McEliece assumptions using the techniques in [Dowsley et al. 2008] and [Hemenway et al. 2009]. Perhaps the re-randomization techniques in [David et al. 2010] can also be applied to obtain a similar primitive.

3.2. A construction based on Re-Randomization

We now recall a construction of a IND-CPA secure statistically (resp. computationally) lossy cryptosystem from a statistically (resp. computationally) re-randomizable cryptosystem which is given and proven in [Hemenway et al. 2009]. Furthermore, we show that it is possible to construct a public key distinguishing algorithm for this construction.

A cryptosystem is called statistically (resp. computationally) re-randomizable if, given a ciphertext c and a public key, it is possible to re-randomize c obtaining a new valid chipertext c' which encrypts the same plain-text message while being statistically (resp. computationally) indistinguishable from the original c. Although different definitions of re-randomizable cryptosystems exist, we consider a definition similar to the one given in [Hemenway et al. 2009].

Notice that it is possible to obtain re-randomizable cryptosystems from homomorphic cryptosystems, DDH, q-power DDH, q-strong Diffie-Hellman and Bilinear Diffie-Hellman. Hence, our construction unifies all previous constructions of fully simulatable oblivious transfer, which are based on these assumptions and also on smooth projective hashing (that also yields lossy encryption).

Definition 3. *Let* $(\mathtt{Gen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{ReRand})$ *be a statistically (resp. computationally) re-randomizable IND-CPA secure public-key cryptosystem, we create statistically (resp. computational)* $(\overline{G}_{inj}, \overline{G}_{lossy}, \overline{E}, \overline{D})$ *as follows:*

- ***Key Generation***: $\overline{\mathsf{G}}(1^\lambda, \mathsf{inj})$ *generates a pair* $(pk, sk) \leftarrow \mathtt{Gen}(1^\lambda)$. *Then* $\overline{\mathsf{G}}(1^\lambda, \mathsf{inj})$ *generates* $K_0 = \mathtt{Enc}(pk, 0)$, $K_1 = \mathtt{Enc}(pk, 1)$. $\overline{\mathsf{G}}(1^\lambda, \mathsf{inj})$ *returns* $(pk, sk) = ((pk, K_0, K_1), sk)$.
  $\overline{\mathsf{G}}(1^\lambda, \mathsf{lossy})$ *runs* $\mathtt{Gen}(1^\lambda)$, *generating a pair* (pk, sk). *Then, it generates* $K_0 = \mathtt{Enc}(pk, 0)$, $K_1 = \mathtt{Enc}(pk, 0)$. $\overline{\mathsf{G}}(1^\lambda, \mathsf{lossy})$ *returns* $(pk, sk) = ((pk, K_0, K_1), sk)$.
- ***Encryption***: $\overline{\mathtt{E}}(pk, b) = \mathtt{ReRand}(pk, K_b)$ *for* $b \in \{0, 1\}$
- ***Decryption***: $\overline{\mathtt{D}}(sk, c)$, *simply outputs* $\mathtt{Dec}(sk, c)$.

An algorithm that distinguishes lossy public keys from injective public keys given the corresponding secret key can be constructed as follows:

- $\mathtt{KD}(sk, pk)$: First computes test ciphertext $c = \mathtt{E}(pk, 1)$. Then output whatever $\mathtt{D}(sk, c)$ outputs.

It is clear that, if the public key pk is injective, this algorithm will output 1, which is the information encrypted into the ciphertext. Otherwise, if the public key is lossy, this algorithm will output 0, since the ciphertext generated by $\mathtt{E}$ is always an encryption of 0 if the public key pk is lossy. Thus, the proposed algorithm $\mathtt{KD}$ successfully distinguishes lossy and injective public keys given the corresponding secret key.

4. The Protocol

The protocol introduced in this section was inspired by the fully simulatable protocol for Oblivious Transfer under the DDH assumptions presented in [Lindell 2008]. In this protocol, the sender (Alice) inputs a pair of bits b_0, b_1 and the receiver (Bob) inputs a *choice bit* σ, Bob receives the bit b_σ and Alice receives nothing ($\perp$). In the end of the protocol Bob must not have learnt anything about the other bit $b_{1-\sigma}$ and Alice must not have learnt anything about Bob's choice bit σ.

Apart from the IND-CPA secure lossy cryptosystem $(\mathtt{Gen}, \mathtt{Enc}, \mathtt{Dec})$, we also assume the existence of a perfectly hiding commitment scheme Com_h and a perfectly binding commitment scheme Com_b. Notice that such commitments can be obtained from the DDH assumptions (and its variations). Moreover, the smooth projective hashing and homomorphic encryption based constructions also rely on such commitment schemes. Thus, our construction unifies the previous fully simulatable oblivious transfer protocols based on such assumptions.

The protocol is secure against *static* malicious adversaries, in other words, the parties may deviate from the protocol but must have their behavior fixed before the execution begins, behaving maliciously or honestly during the whole execution.

1. For $i = 1, \ldots, \ell$, the receiver Bob chooses a random bit $\sigma_i \in_R \{0, 1\}$ and runs $\mathtt{G}(1^n, \mathsf{inj})$, obtaining ℓ injective key pairs (pk_i^{inj}, sk_i^{inj}). It also runs $\mathtt{G}(1^n, \mathsf{lossy})$, obtaining ℓ lossy key pairs $(pk_i^{lossy}, sk_i^{lossy})$.For each bit σ_i, Bob generates a *pair of public keys* $(\gamma_i^{\sigma_i}, \gamma_i^{1-\sigma_i})$ such that $\gamma_i^{\sigma_i} = pk_i^{inj}$ and $\gamma_i^{1-\sigma_i} = pk_i^{lossy}$. Bob sends all of the pairs $\langle(\gamma_1^0, \gamma_1^1), \ldots, (\gamma_\ell^0, \gamma_\ell^1)\rangle$ to Alice.
2. Coin tossing:
   (a) Alice chooses a random $s \in_R \{0, 1\}^\ell$ and sends $\mathsf{Com}_h(s)$ to Bob.
   (b) Bob chooses a random $s' \in_R \{0, 1\}^\ell$ and sends $\mathsf{Com}_b(s')$ to Bob.
   (c) Alice and Bob send decommitments to $\mathsf{Com}_h(s)$ and $\mathsf{Com}_b(s')$ respectively, and set $r = s \oplus s'$. Denote $r = r_1, \ldots, r_\ell$.

3. For every i for which $r_i = 1$, Bob sends $(sk_i^{inj}, sk_i^{lossy})$ to Alice. In addition, for every j for which $r_j = 0$, Bob sends a "reordering" of γ_j^0 and γ_j^1 such that, in the resulting tuples (γ_j^0, γ_j^1), γ_j^σ is an injective public key and $\gamma_j^{1-\sigma}$ is a lossy public key. This reordering is a bit such that if it equals 0 then the tuples are left as is, and if it equals 1 then γ_j^0 and γ_j^1 are interchanged.
4. Alice checks that, for every i for which $r_i = 1$ it received a valid secret key pair $(sk_i^{inj}, sk_i^{lossy})$, such that exactly one of the corresponding public keys is injective and exactly one is lossy. Furthermore, it checks that exactly one of the public keys (γ_i^0, γ_i^1) received is injective and exactly one of the public keys is lossy by running $\mathtt{KD}(sk_i^{inj}, \gamma_i^0)$ and $\mathtt{KD}(sk_i^{inj}, \gamma_i^1)$. If any of the checks fail, Alcie halts and outputs $\perp$. Otherwise it proceeds to the next step.
5. For each j for which $r_j = 0$ denote each (γ_j^0, γ_j^1) as $(\Upsilon_n^0, \Upsilon_n^1)$ for $n = 1, \ldots, \ell'$, where ℓ' is the total number of j for which $r_j = 0$. Employing a reduction given in [Damgård et al. 1999], Alice chooses n random bits $b_{0,1}, \ldots, b_{0,n}$ and n random bits $b_{1,1}, \ldots, b_{1,n}$ such that $b_0 = b_{0,1} \oplus \ldots \oplus b_{0,n}$ and $b_1 = b_{1,1} \oplus \ldots \oplus b_{1,n}$.

   For each pair of bits $b_{0,n}, b_{1,n}$ and each $(\Upsilon_n^0, \Upsilon_n^1)$ Alice computes a random bit $\mu_n \in_R \{0, 1\}$ and the encryption of $b_{\bullet,n} \oplus \mu_n$ for each bit in the pair, obtaining $\hat{b}_{0,n} = \mathtt{E}(\Upsilon_n^0, b_{0,n} \oplus \mu_n)$ and $\hat{b}_{1,n} = \mathtt{E}(\Upsilon_n^1, b_{1,n} \oplus \mu_n)$. Alice sends the bits μ_n and the pairs $(\hat{b}_n^0, \hat{b}_n^1)$ to Bob.
6. For each pair of bit $\hat{b}^{\sigma,n}$ and bit μ_n Bob computes $b_{\sigma,n} = \mathtt{D}(sk_n^{inj}, \hat{b}_{\sigma,n}) \oplus \mu_n$. Finally, bob computes $b_\sigma = b_{\sigma,1} \oplus \ldots \oplus b_{\sigma,n}$, obtaining b_σ.

Correctness: Before proceeding to the proof of security, we show that the protocol above is correct, in the sense that, if both Alice and bob are honest, the correct output is obtained. First, observe that in the reordered pairs obtained after the coin tossing, Υ_n^σ is an injective key, enabling an honest Bob to extract a bit encrypted with it (*i.e.*, $b = \mathtt{D}(sk_n^{inj}, \mathtt{E}(\Upsilon_n^\sigma, b))$). However, the keys $\Upsilon_n^{1-\sigma}$ are lossy, which makes it impossible for Bob to obtain the value of a bit encrypted with those keys. Also, since $\hat{b}_{\sigma,n} = \mathtt{E}(\Upsilon_n^\sigma, b_{\sigma,n} \oplus \mu_n)$ for a random value of μ_n, Bob is not able to obtain the original value of $b_{\sigma,n}$ without first obtaining the corresponding μ_n.

Given that Alice and Bob are honest, it is possible for Bob to obtain the bit b_σ since, based on the facts stated above, it is possible to obtain the value of each bit $b_{\sigma,n}$ computing $b_{\sigma,n} = \mathtt{D}(sk_n^{inj}, \hat{b}_{\sigma,n}) \oplus \mu_n$ after receiving the correct values of μ_n and $\hat{b}_{\sigma,n}$ from Alice. In order to obtain the original bit b_σ, Bob employs the reduction given and proven in [Damgård et al. 1999] computing $b_\sigma = \oplus_{i=1}^n b_{\sigma,i}$, correctly yielding: $b_\sigma = (b_{\sigma,1} \oplus \mu_1) \oplus \ldots \oplus (b_{\sigma,n} \oplus \mu_n)$.

Notice that, if statistically lossy encryption is employed, the resulting protocol offers statistical security for the sender, since the ciphertexts $\hat{b}_{1-\sigma,n}$ *statistically* loose information about the bits corresponding to $\hat{b}_{1-\sigma}$. On the other hand, if computationally lossy encryption is employed, the resulting protocol offers computational security for the sender, since the ciphertexts $\hat{b}_{1-\sigma,n}$ computationally loose information about the bits corresponding to $\hat{b}_{1-\sigma}$. The security for the receiver is computational in both cases, since it relies on the computational indistinguishability of lossy and injective keys.

4.1. Simulator for the case Alice (sender) is corrupted

In order to prove the security of the proposed protocol we adapt the simulators given in [Lindell 2008] for the case where the sender is corrupted and the case the receiver is corrupted. Notice that the resulting simulators have the same running time of the simulators in [Lindell 2008], since the steps involved are essentially the same. Let $\mathcal{A}_1$ be a non-uniform probabilistic polynomial-time real adversary that controls Alice. We construct a non-uniform probabilistic expected polynomial-time ideal-model adversary/simulator $\mathcal{S}_1$. $\mathcal{S}_1$ uses rewinding in order to ensure that all of the "checked" public key pairs are valid (*i.e.*,exactly one of them is lossy), whereas both keys contained in the "unchecked" public key pairs are injective. This enables it to obtain both messages input by $\mathcal{A}_1$ into the protocol. $\mathcal{S}_1$ then sends these inputs to the trusted party, and the honest party Bob in the ideal model will receive the same message that it would have received in a real execution with $\mathcal{A}_1$ (or more accurately, a message that is computationally indistinguishable from that message).

We now describe $\mathcal{S}_1$ formally. Upon input 1^n and (b_0, b_1), the machine $\mathcal{S}_1$ invokes $\mathcal{A}_1$ upon the same input and works as follows:

1. $\mathcal{S}_1$ chooses a random $r \in_R 0, 1^\ell$ and generates public key pairs $(\gamma_1^0, \gamma_1^1), \ldots, (\gamma_\ell^0, \gamma_\ell^1)$ with the following property:
   (a) For every i for which $r_i = 1$, $\mathcal{S}_1$ constructs (γ_i^0 and γ_i^1) like an honest Bob. It runs $\mathtt{G}(1^n, \mathsf{inj})$, obtaining ℓ injective key pairs (pk_i^{inj}, sk_i^{inj}). It also runs $\mathtt{G}(1^n, \mathsf{lossy})$, obtaining ℓ lossy key pairs $(pk_i^{lossy}, sk_i^{lossy})$. $\mathcal{S}_1$ generates a *pair of public key* $(\gamma_i^{\sigma_i}, \gamma_i^{1-\sigma_i})$ such that $\gamma_i^{\sigma_i} = pk_i^{inj}$ and $\gamma_i^{1-\sigma_i} = pk_i^{lossy}$, for random bits $\sigma_i \in_R \{0, 1\}$.
   (b) For every j for which $r_j = 0$, $\mathcal{S}_1$ constructs (γ_j^0, γ_j^1) such that both γ_j^0 and γ_j^1 are injective keys.

   $\mathcal{S}_1$ hands the public key pairs to $\mathcal{A}_1$.
2. Simulation of the coin tossing: $\mathcal{S}_1$ simulates the coin tossing so that the result is r, as follows:
   (a) $\mathcal{S}_1$ receives a commitment c_h from $\mathcal{A}_1$.
   (b) $\mathcal{S}_1$ chooses a random $s' \in_R \{0, 1\}^\ell$ and hands $c_b = \mathsf{Com}_h(s')$ to $\mathcal{A}_1$.
   (c) If $\mathcal{A}_1$ does not send a valid decommitment to c_h, then $\mathcal{S}_1$ simulates Bob aborting and sends $\perp$ to the trusted party. Then $\mathcal{S}_1$ outputs whatever $\mathcal{A}_1$ outputs and halts. Otherwise, let s be the decommitted value. $\mathcal{S}_1$ proceeds as follows:
      i. $\mathcal{S}_1$ sets $s' = r \oplus s$, rewinds $\mathcal{A}_1$, and hands it $\mathsf{Com}_b(s')$.
      ii. If $\mathcal{A}_1$ decommits to s, then $\mathcal{S}_1$ proceeds to the next step. If $\mathcal{A}_1$ decommits to a value $\tilde{s} \neq s$, then $\mathcal{S}_1$ outputs fail. Otherwise, if it does not decommit to any value, $\mathcal{S}_1$ returns to the previous step and tries again until $\mathcal{A}_1$ does decommit to s. (We stress that in every attempt, $\mathcal{S}_1$ hands $\mathcal{A}_1$ a commitment to the same value s'. However, the randomness used to generate the commitment $\mathsf{Com}_b(s')$ is independent each time.)[1]

[1] Similarly to the DDH based protocol of [Lindell 2008], this strategy by $\mathcal{S}_1$ does not actually guarantees that it runs in expected polynomial-time. Fortunately this issue is solved in [Lindell 2008] and we refer the reader to that work for detailed information.

3. Upon receiving a valid decommitment to s from $\mathcal{A}_1$, simulator $\mathcal{S}_1$ decommits to $\mathcal{A}_1$, revealing s'. (Note that $r = s \oplus s'$.)
4. For every i for which $r_i = 1$, simulator $\mathcal{S}_1$ hands $\mathcal{A}_1$ the secret key pairs $(sk_i^{inj}, sk_i^{lossy})$ correspondent to the public keys (γ_i^0, γ_i^1). In addition, $\mathcal{S}_1$ hands $\mathcal{A}_1$ a random reordering of the pairs (γ_j^0, γ_j^1) for every j for which $r_j = 0$.
5. If $\mathcal{A}_1$ does not reply with a valid message, then $\mathcal{S}_1$ sends $\perp$ to the trusted party, outputs whatever $\mathcal{A}_1$ outputs and halts. Otherwise, it receives the bits μ_n and a series of pairs $(\hat{b}_n^0, \hat{b}_n^1)$. $\mathcal{S}_1$ then follows the instructions of Bob for obtaining both b_0 and b_1. Unlike an honest Bob, it decrypts both $\hat{b}_n^0$ and $\hat{b}_n^1$ with the injective secret keys corresponding to $(\Upsilon_n^0, \Upsilon_n^1)$, obtaining a series of pairs $(b_{0,n}, b_{1,n})$. It then computes $b_0 = (b_{0,1} \oplus \mu_1) \oplus \ldots \oplus (b_{0,n} \oplus \mu_n)$ and $b_1 = (b_{1,1} \oplus \mu_1) \oplus \ldots \oplus (b_{1,n} \oplus \mu_n)$. $\mathcal{S}_1$ sends the pair (b_0, b_1) to the trusted party as the first party's input, outputs whatever $\mathcal{A}_1$ outputs and halts.

Theorem 4. *The joint output distribution of $\mathcal{S}_1$ and an honest Bob in an ideal execution is computationally indistinguishable from the output distribution of $\mathcal{A}_1$ and an honest Bob in a real execution.*

Proof. In order to prove this theorem we adapt the proof given in [Lindell 2008]. Notice that the view of $\mathcal{A}_1$ in the simulation with $\mathcal{S}_1$ is indistinguishable from its view in a real execution. The sole difference in this view is due to the fact that the public keys γ_j^0 and γ_j^1 for which $r_j = 0$ are both injective public keys.

The only other difference would be in the coin tossing phase (and the rewinding). However, since the commitment sent by $\mathcal{A}_1$ is binding and since Bob generates its commitment after receiving $\mathcal{A}_1$'s commitment, it is clear that the result of the coin tossing in a real execution and in the simulation with $\mathcal{S}_1$ are statistically close to uniform (where the only difference is due to the negligible probability that $\mathcal{A}_1$ will break the computational binding property of the commitment scheme.) In the simulation by $\mathcal{S}_1$, the outcome is always uniformly distributed, assuming that $\mathcal{S}_1$ does not output fail. Since $\mathcal{S}_1$ outputs fail when $\mathcal{A}_1$ breaks the computational binding of the commitment scheme, this occurs with at most negligible probability (a rigorous analysis is given in [Goldreich and Kahan 1996]). Thus, the joint distribution of the coin tossing results in a real execution and in the simulation with $\mathcal{S}_1$ are statistically close.

Therefore, the only remaining difference lies in the generation of public keys γ_j^0 and γ_j^1. Indistinguishability follows intuitively from the definition of lossy encryption (*i.e.* lossy public keys are computationally indistinguishable from injective public keys). This is formally proven by constructing a machine D that distinguishes many injective keys from many lossy keys, which implies in breaking the lossy key indistinguishability property of the lossy cryptosystem. D receives a set of public keys and runs in exactly the same way as $\mathcal{S}_1$ but constructs the γ_j^0 and γ_j^1 public keys (for which $r_j = 0$) in such a way that one is injective and the other is from its input, in random order. Furthermore, it provides the reordering so that all of the injective keys it generates are associated with σ and all of the ones it receives externally are associated with $1 - \sigma$ (we assume that D is given the input σ of Bob). Note that, if D receives a set of injective keys, then the view of $\mathcal{A}_1$ is exactly the same as in the simulation with $\mathcal{S}_1$ (because all the keys are injective). Otherwise, if D receives a set of lossy keys, then the view of $\mathcal{A}_1$ is exactly the same as in a real execution (because only the keys associated with σ are injective). This shows

that the output of $\mathcal{A}_1$ in a real execution and the output of $\mathcal{S}_1$ in an ideal execution are indistinguishable (recall that $\mathcal{S}_1$ outputs whatever $\mathcal{A}_1$ outputs).

However,it is necessary to show this for the joint distribution of the output of $\mathcal{A}_1$ (or $\mathcal{S}_1$) and an honest Bob. First, recall that Bob receives m_σ as output, where σ is the honest Bob's input. Next, assume that there exists a polynomial-time distinguisher D' that distinguishes between the real and ideal distributions with non-negligible probability. To complete this proof we construct another distinguisher D that distinguishes injective keys from lossy keys. D receives Bob's input σ and a set of keys that are either injective or lossy. D then works exactly as above (*i.e.*, constructing the public keys γ_j^0 and γ_j^1 so that in the reordering step, all the γ_j^σ keys are those it generated itself and all the $\gamma_j^{1-\sigma}$ tuples are those it received as input). D is able to decrypt each $\hat{b}_{\sigma,n}$ and obtain m_σ, since it generated all of the γ_j^σ keys. Machine D then does this, and runs D' on the output of $\mathcal{A}_1$ and the message m_σ (which is the output that an honest Bob would receive). Finally, D outputs whatever D' does. If D receives lossy keys, then the output distribution generated is exactly the same of a real execution between $\mathcal{A}_1$ and Bob. On the contrary, if it receives injective keys, the output distribution is exactly the same of an ideal execution with $\mathcal{S}_1$. (Notice that the distribution over the γ public keys generated by D with knowledge of σ is identical to the distribution generated by $\mathcal{S}_1$ without knowledge of σ. The reason for this is that when all the keys are injective, their ordering makes no difference.) We conclude that D distinguishes lossy and injective public keys with non-negligible probability, in contradiction to the definition of lossy encryption. Thus, the REAL and IDEAL output distributions are computationally indistinguishable.

The last step is to prove that $\mathcal{S}_1$ runs in expected polynomial-time. However, as in the protocols given in [Lindell 2008] this is not true. Fortunately, this can be fixed by a direct application of the techniques proposed in [Lindell 2008] and [Goldreich and Kahan 1996], and we refer the reader to these works for a detailed analysis. It is shown that these techniques yield a simulator that is guaranteed to run in expected polynomial time. Furthermore, the output of the simulator is only negligibly far from the original (simplified) strategy described in this work. Thus, after applying these techniques, our simulator runs in expected polynomial time, with the result being that the output in a simulation is only negligibly different from the output in a real execution. □

4.2. Simulator for the case Bob (receiver) is corrupted

Once again we base our simulator and proof on the techniques proposed in [Lindell 2008]. Let $\mathcal{A}_2$ be any non-uniform probabilistic polynomial-time adversary controlling Bob, we construct a non-uniform probabilistic expected polynomial-time simulator $\mathcal{S}_2$. The simulator $\mathcal{S}_2$ extracts the bit σ used by $\mathcal{A}_2$ by rewinding it and obtaining the reordering of public keys that it had previously opened. Formally, upon input 1^n and σ, the simulator $\mathcal{S}_2$ invokes $\mathcal{A}_2$ upon the same input and works as follows:

1. $\mathcal{S}_2$ receives a series of public key pairs $(\gamma_1^0, \gamma_1^1), \ldots, (\gamma_\ell^0, \gamma_\ell^1)$ from $\mathcal{A}_2$.
2. $\mathcal{S}_2$ hands $\mathcal{A}_2$ a commitment $c_h = \mathsf{Com}_h(s)$ to a random $s \in_R \{0,1\}^\ell$, receives back c_b, decommits to c_h and receives $\mathcal{A}_2$'s decommitment to c_b. $\mathcal{S}_2$ then receives all of the sk_i keys from $\mathcal{A}_2$, for i where $r_i = 1$, and the reorderings for j where $r_j = 0$. If the pairs (γ_i^0, γ_i^1) sent by $\mathcal{A}_2$ are not valid (as checked by Alice in the

protocol) or $\mathcal{A}_2$ did not send valid decommitments, $\mathcal{S}_2$ sends $\perp$ to the trusted party, outputs whatever $\mathcal{A}_2$ outputs, and halts. Otherwise, it continues to the next step.

3. $\mathcal{S}_2$ rewinds $\mathcal{A}_2$ back to the beginning of the coin-tossing, hands $\mathcal{A}_2$ a commitment $\tilde{c}_h = \mathsf{Com}_h(\tilde{s})$ to a fresh random $\tilde{s} \in_R \{0,1\}^\ell$, receives back some $\tilde{c}_b$, decommits to $\tilde{c}_h$ and receives $\mathcal{A}_2$'s decommitment to $\tilde{c}_b$. In addition, $\mathcal{S}_2$ receives the $(sk_i^{inj}, sk_i^{lossy})$ secret key pairs and reorderings. If any of the pairs (γ_i^0, γ_i^1) are not valid, $\mathcal{S}_2$ repeats this step using fresh randomness each time, until all pairs are valid.
4. Following this, $\mathcal{S}_2$ rewinds $\mathcal{A}_2$ to the beginning and resends the exact messages of the first coin tossing (resulting in exactly the same transcript as before).
5. Denote by r the result of the first coin tossing (Step 2 above), and $\tilde{r}$ the result of the second coin tossing (Step 3 above). If $r = \tilde{r}$ then $\mathcal{S}_2$ outputs fail and halts. Otherwise, $\mathcal{S}_2$ searches for a value t such that $r_t = 0$ and $\tilde{r}_t = 1$. (Note that by the definition of the simulation, exactly one of γ_t^0 and γ_t^1 is injective. Otherwise, the values would not be considered valid.) If no such t exists (*i.e.*, for every t such that $r_t \neq \tilde{r}_t$ it holds that $r_t = 1$ and $\tilde{r}_t = 0$),then $\mathcal{S}_2$ begins the simulation from scratch with the exception that it must find r and $\tilde{r}$ for which all values are valid (*i.e.*, if for r the values sent by $\mathcal{A}_2$ are not valid it does not terminate the simulation but rather rewinds until it finds an r for which the responses of $\mathcal{A}_2$ are all valid).
   If $\mathcal{S}_2$ does not start again, we have that it has sk_t and can determine which of $(\gamma_t^0$ and γ_t^1 is injective. Furthermore, since $\tilde{r}_t = 1$, the reordering that $\mathcal{S}_2$ receives from $\mathcal{A}_2$ after the coin tossing indicates whether the public key pair is associated with 0 (if γ_t^0 is injective) or 1 (if γ_t^1 is injective) . $\mathcal{S}_2$ sets $\sigma = 0$ if after the reordering γ_t^0 is injective, and sets $\sigma = 1$ if after the reordering γ_t^1 is injective. (Note that exactly one of the keys is injective because this is checked in the second coin tossing.)
6. $\mathcal{S}_2$ sends σ to the trusted party and receives back a bit $b = b'_\sigma$. Simulator $\mathcal{S}_2$ then computes the last message from Alice to Bob honestly, setting $b_\sigma = b$, $b_{1-\sigma} \in_R \{0,1\}$ and running the instruction used by an honest Alice to compute the last message. $\mathcal{S}_2$ hands $\mathcal{A}_2$ these messages and outputs whatever $\mathcal{A}_2$ outputs and halts.

Theorem 5. *The output distribution of $\mathcal{A}_2$ in a real execution with an honest Alice (with input (m_0, m_1)) is computationally indistinguishable from the output distribution of $\mathcal{S}_2$ in an ideal execution with an honest Alice (with the same input (m_0, m_1))*

Proof. First, notice that $\mathcal{S}_2$ outputs fail with probability at most $2^{1-\ell}$ even if $r = \tilde{r}$ in later rewindings, which may occur if $\mathcal{S}_2$ has to start again from scratch. A detailed analysis of this probability is given in [Lindell 2008]. Given this fact, we proceed to show indistinguishability of the ideal and real executions adapting the proof of [Lindell 2008].

Notice that, if $\mathcal{S}_2$ does not output fail, $\mathcal{A}_2$ views a final transcript consisting of the first coin tossing (that is distributed exactly as in a real execution) and the last message from $\mathcal{S}_2$ to $\mathcal{A}_2$. This message is not honestly generated, since c_σ is indeed an encryption of m_σ, but $c_{1-\sigma}$ is actually an encryption of an arbitrary value (which is not necessarily of $m_{1-\sigma}$). However, it follows from the definition of lossy encryption (specifically from the lossiness property) that, for any lossy public key $\gamma_j^{1-\sigma}$, the value encrypted in $\hat{b}_{1-\sigma,n}$ is at least computationally indistinguishable from a random value in the lossy cryptosystem's plaintext space. This implies that the distribution of values $\hat{b}_{1-\sigma,n}$ generated under a lossy key from a random plaintext value is computationally indistinguishable from the distribution of values $\hat{b}_{1-\sigma,n}$ generated from the values $m_{1-\sigma}$. Thus, $\mathcal{A}_2$'s view in the execution

with $\mathcal{S}_2$ is at least computationally indistinguishable from its view in a real execution with Alice (the only difference being if $\mathcal{S}_2$ outputs fail).

Note that, if statistically lossy encryption is used, the values $\hat{b}_{1-\sigma,n}$ are uniformly distributed. Thus, $\mathcal{A}_2$'s view in the execution with $\mathcal{S}_2$ is statistically close to its view in a real execution with Alice (the only difference being if $\mathcal{S}_2$ outputs fail).

It remains to prove that $\mathcal{S}_2$ runs in expected polynomial-time, a fact that follows directly from the analysis in [Lindell 2008].

□

5. Conclusion

In this paper we propose a general construction of efficient fully simulatable oblivious transfer based on lossy encryption. Our construction can be realized from a multitude of underlying primitives and computational assumptions such as smooth projective hashing, re-randomization, factorization, discrete logarithm and coding theory problems. Additionally, the proposed protocol essentially unifies known efficient fully simulatable OT protocols in the plain model. Furthermore, this protocol completes the proof that several flavors of lossy encryption are equivalent to fully simulatable oblivious transfer, since the converse is proved in [Lindell 2008] for smooth projective hashing and re-randomization based constructions.

In order to obtain our results we introduce the primitive of computationally lossy encryption, which may be of independent interest to other applications. In this case, it can be used to obtain a construction of efficient fully simulatable OT based on the McEliece assumptions, which can be shown to realize computationally lossy encryption using the techniques of [Dowsley et al. 2008]. However, this construction would still be based on the assumption that a perfectly hiding commitment scheme and a perfectly binding commitment scheme exist.

Apart from unveiling new theoretical relations between cryptographic primitives, our contributions also provide a general framework for constructing efficient fully simulatable oblivious transfer protocols, which are a central building block in two- and multi-party computation. However, we have not yet achieved security in the universal composability framework. As a future work we suggest the creation a of a general unifying framework for universally composable oblivious transfer realizable under the same underlying computational assumptions as our fully simulatable construction. Moreover, we point out further investigation of applications for computationally lossy encryption.

References

Barak, B. and Lindell, Y. (2004). Strict polynomial-time in simulation and extraction. *SIAM J. Comput.*, 33(4):738–818.

Bellare, M., Hofheinz, D., and Yilek, S. (2009). Possibility and impossibility results for encryption and commitment secure under selective opening. In *Proceedings of the 28th Annual International Conference on Advances in Cryptology: the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '09, pages 1–35, Berlin, Heidelberg. Springer-Verlag.

Camenisch, J., Neven, G., and Shelat, A. (2007). Simulatable adaptive oblivious transfer. In Naor, M., editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 573–590. Springer.

Damgård, I., Kilian, J., and Salvail, L. (1999). On the (im)possibility of basing oblivious transfer and bit commitment on weakened security assumptions. In *EUROCRYPT'99: Proceedings of the 17th international conference on Theory and application of cryptographic techniques*, pages 56–73, Berlin, Heidelberg. Springer-Verlag.

David, B. M., Nascimento, A. C. A., and Nogueira, R. B. (2010). Oblivious transfer based on the mceliece assumptions with unconditional security for the sender. In *X Simposio Brasileiro de Segurança da Informação e de Sistemas Computacionais*.

Dowsley, R., van de Graaf, J., Müller-Quade, J., and Nascimento, A. C. A. (2008). Oblivious transfer based on the mceliece assumptions. In Safavi-Naini, R., editor, *ICITS*, volume 5155 of *Lecture Notes in Computer Science*, pages 107–117. Springer.

Goldreich, O. and Kahan, A. (1996). How to construct constant-round zero-knowledge proof systems for np. *Journal of Cryptology*, 9:167–189. 10.1007/BF00208001.

Green, M. and Hohenberger, S. (2007). Blind identity-based encryption and simulatable oblivious transfer. In Kurosawa, K., editor, *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 265–282. Springer.

Hemenway, B., Libert, B., Ostrovsky, R., and Vergnaud, D. (2009). Lossy encryption: Constructions from general assumptions and efficient selective opening chosen ciphertext security. Cryptology ePrint Archive, Report 2009/088. `http://eprint.iacr.org/`.

Kilian, J. (1988). Founding crytpography on oblivious transfer. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 20–31, New York, NY, USA. ACM.

Lindell, A. Y. (2008). Efficient fully-simulatable oblivious transfer. In *Proceedings of the 2008 The Cryptopgraphers' Track at the RSA conference on Topics in cryptology*, CT-RSA'08, pages 52–70, Berlin, Heidelberg. Springer-Verlag.

Peikert, C., Vaikuntanathan, V., and Waters, B. (2008). A framework for efficient and composable oblivious transfer. In Wagner, D., editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571. Springer Berlin / Heidelberg.

Rabin, M. O. (1981). How to exchange secrets by oblivious transfer. Technical Memo TR-81, Aiken Computation Laboratory, Harvard University.