How to Simulate It in Isabelle: Towards Formal Proof for Secure Multi-Party Computation

How to Simulate It in Isabelle: Towards Formal Proof for Secure Multi-Party Computation

David Butler[1] and David Aspinall[1] and Adrià Gascón[2]

[1] The Alan Turing Institute and University of Edinburgh
[2] The Alan Turing Institute and University of Warwick

Abstract. In cryptography, secure Multi-Party Computation (MPC) protocols allow participants to compute a function jointly while keeping their inputs private. Recent breakthroughs are bringing MPC into practice, solving fundamental challenges for secure distributed computation. Just as with classic protocols for encryption and key exchange, precise guarantees are needed for MPC designs and implementations; any flaw will give attackers a chance to break privacy or correctness. In this paper we present the first (as far as we know) formalisation of some MPC security proofs. These proofs provide probabilistic guarantees in the computational model of security, but have a different character to machine proofs and proof tools implemented so far — MPC proofs use a *simulation* approach, in which security is established by showing indistinguishability between execution traces in the actual protocol execution and an ideal world where security is guaranteed by definition. We show that existing machinery for reasoning about probabilistic programs can be adapted to this setting, paving the way to precisely check a new class of cryptography arguments. We implement our proofs using the CryptHOL framework inside Isabelle/HOL.

Keywords: oblivious transfer, cryptography, simulation-based proof, formal verification

1 Introduction

Correctness guarantees are essential for cryptographic protocols and it is an area where formalisation continues to have impact. Older work was restricted to the *symbolic (Dolev-Yao) model* [11], where cryptographic primitives are modelled as abstract operations and assumed to be unbreakable. The symbolic model provides a baseline for correctness but modern cryptography is based on the more realistic *computational model* [1]. Adversaries are now allowed to break primitives, but are assumed to have limited computational power — typically, polynomial time in a security parameter n, such as a key size. Proofs in the computational model provide probabilistic guarantees: an adversary can break a security property only with negligible probability, i.e. probability bounded

This work was supported by The Alan Turing Institute under the EPSRC grant EP/N510129/1.

by a negligible function $\mu(n)$. There are two main proof styles, the *game-based* approach [21] and the *simulation-based* approach sometimes called the real/ideal world paradigm [14].

The simulation-based approach is a general proof technique especially useful for arguing about security of Multi-Party Computation (MPC) protocols. MPC is an area of cryptography concerned with enabling multiple parties to jointly evaluate a public function on their private inputs, without disclosing unnecessary information (that is, without leaking any information about their respective inputs that cannot be deduced from their sizes or the result of the computation). Several generic techniques can be used for that goal including Yao's garbled circuits [15, 22], the GMW protocol [12], and other protocols based on secret-sharing [8,16]. These differ in whether they are designed for an arbitrary or fixed number of parties, how the computed function is represented (e.g, Boolean vs. arithmetic circuits), which functions can be represented (e.g, bounded-degree polynomials vs. arbitrary polynomials), as well trade-offs regarding communication, computation requirements, and security guarantees.

In the last decade, groundbreaking developments have brought MPC closer to practice. Efficient implementations of the protocols listed above are available [9, 13, 17, 23], and we are now seeing the beginning of general solutions to fundamental security challenges of distributed computation. Security in these settings is proved by establishing a simulation between the *real world*, where the protocol plays out, and an *ideal world*, which is taken as the definition of security. This formalises the intuition that a protocol is secure if it can be simulated in an ideal environment in which there is no data leakage by definition.

A central protocol in MPC is Oblivious Transfer (OT), which allows a *sender* to provide several values and a *receiver* to choose some of them to receive, without learning the others, and without the sender learning which has been chosen. In this paper we build up to a security proof of the Naor-Pinkas OT [19], a practically important 1-out-of-2 oblivious transfer protocol (the receiver chooses one out of two messages). This can be used as a foundation for more general MPC, as secure evaluation of arbitrary circuits can be based on OT [12].

Contribution. As far as we know, this is the first formalisation of MPC proofs in a theorem prover. Our contributions are as follows.

- Starting from the notion of computational indistinguishablity, we formalise the simulation technique following the general form given by Lindell [14].
- Lindell's method spells out a process but leaves details of reasoning to informal arguments in the cryptographer's mind; to make this fully rigorous, we use probabilistic programs to encode *views* of the real and ideal worlds which can be successively refined to establish equivalence. This is a general method which can be followed for other protocols and in other systems; it corresponds to *hybrid arguments* often used in cryptography.
- As examples of the method, we show information-theoretic security for a two-party secure multiplication protocol that uses a trusted initialiser, and a proof of security in the semi-honest model of the Naor-Pinkas OT protocol.

The latter involves a reduction to the DDH assumption (a computational hardness assumption).
– Finally, we demonstrate how a formalisation of security of a 1-out-of-2 OT can be extended to formalising the security of an AND gate.

We build on Andreas Lochbihler's recent *CryptHOL* framework [18], which provides tools for encoding probabilistic programs using a shallow embedding inside Isabelle/HOL. Lochbihler has used his framework for game-based cryptographic proofs, along similar lines to proofs constructed in other theorem provers [2, 20] and dedicated tools such as EasyCrypt [3].

Outline. In Sect. 2 we give an overview of the key parts of CryptHOL that we use and extend. Sect. 3 shows how we define computational indistinguishability in Isabelle and Sect. 4 shows how it is used to define simulation-based security. In Sect. 4.1 we demonstrate how we use a probabilistic programming framework to do proofs in the simulation-based setting. Sect. 5 gives the proof of security of a secure multiplication protocol as a warm up and Sect. 6 shows the proof of security of the Naor-Pinkas OT protocol. In Sect. 7 we show how an OT protocol can be used to securely compute an AND gate, paving the way towards generalised protocols. Our formalisation is available online at https://github.com/alan-turing-institute/isabelle-mpc.

2 CryptHOL and Extensions

CryptHOL is a probabilistic programming framework based around *subprobability mass functions* (spmfs). An spmf encodes a discrete (sub) probability distribution. More precisely, an spmf is a real valued function on a finite domain that is non negative and sums to at most one. Such functions have type α *spmf* for a domain which is a set of elements of type α. We use the notation from [18] and let $p!x$ denote the subprobability mass assigned by the spmf p to the event x. The weight of an spmf is given by $||p|| = \sum_y p!y$ where the sum is taken over all elementary events of the corresponding type; this is the total mass of probability assigned by the spmf p. If $||p|| = 1$ we say p is *lossless*. Another important function used in our proofs is *scale*. The expression *scale* r p scales, by r, the subprobability mass of p. That is, we have *scale* r $p!x = r.(p!x)$ for $0 \leq r \leq \frac{1}{||p||}$.

Probabilistic programs can be encoded as sequences of functions that compute over values drawn from spmfs. The type α *spmf* is used to instantiate the polymorphic monad operations $return_{spmf} :: \alpha \Rightarrow \alpha\ spmf$ and $bind_{spmf} :: \alpha\ spmf \Rightarrow (\alpha \Rightarrow \beta\ spmf) \Rightarrow \beta\ spmf$.

This gives a shallow embedding for probabilistic programs which we use to define simulations and views, exploiting the monadic do notation. As usual, *do* $\{x \leftarrow p; f\}$ stands for $bind_{spmf}\ p\ (\lambda x.\ do\ f)$.

We note that $bind_{spmf}$ is commutative and constant elements cancel. In particular if p is a lossless spmf, then

$$bind_{spmf}\ p\ (\lambda\_.\ q) = q. \tag{1}$$

Equation 1 can be shown using the lemma *bind_spmf_const*,

$$bind_{spmf}\ p\ (\lambda x.\ q) = scale_{spmf}\ (weight_{spmf}\ p)\ q \tag{2}$$

and the fact *sample_uniform* is lossless and thus has weight equal to one. In Equation 2, $weight_{spmf}\ p$ is $||p||$ described above.

The monad operations give rise to the functorial structure, $map_{spmf} :: (\alpha \Rightarrow \beta) \Rightarrow \alpha\ spmf \Rightarrow \beta\ spmf$.

$$map_{spmf}\ f\ p = bind_{spmf}\ p\ (\lambda x.\ return_{spmf}(f\ x)) \tag{3}$$

CryptHOL provides an operation, $sample\_uniform :: nat \Rightarrow nat\ spmf$ where $sample\_uniform\ n = spmf\_of\_set\ \{.. < n\}$, the lossless spmf which distributes probability uniformly to a set of n elements. Of particular importance in cryptography is the uniform distribution $coin\_spmf = spmf\_of\_set\ \{\mathsf{True}, \mathsf{False}\}$. Sampling from this corresponds to a coin flip.

We also utilise the function $assert\_spmf :: bool \Rightarrow unit\ spmf$ which takes a predicate and only allows the computation to continue if the predicate holds. If it does not hold the current computation is aborted. It also allows the proof engine to pick up on the assertion made.

One way we extend the work of CryptHOL is by adding one time pad lemmas needed in our proofs of security. We prove a general statement given in Lemma 1 and instantiate it prove the one time pads we require.

Lemma 1. *Let f be injective and surjective on $\{.. < q\}$. Then we have*

$$map_{spmf}\ f\ (sample\_uniform\ q) = sample\_uniform\ q.$$

Proof. By definition, $sample\_uniform\ q = spmf\_of\_set\ \{.. < q\}$. Then $map_{spmf}\ f\ (spmf\_of\_set\ \{.. < q\}) = spmf\_of\_set(f\ `\ \{.. < q\})$ follows by simplification and the injective assumption (the infix $`$ is the image operator). Simplification uses the lemma *map_spmf_of_set_inj_on*:

$$inj\_on\ f\ A \implies map_{spmf}\ (spmf\_of\_set\ A) = spmf\_of\_set\ (f\ `\ A).$$

We then have $map_{spmf}\ f\ (spmf\_of\_set\ \{.. < q\}) = spmf\_of\_set(\{.. < q\})$ by using the surjectivity assumption. The lemma then follows from the definition of *sample_uniform*. □

We note a weaker assumption, namely $f\ `\ \{.. < q\} \subseteq \{.. < q\}$ can be used instead of the surjectivity assumption. To complete the proof with this assumption we use the *endo_inj_surj* rule which states

$$finite\ A \implies f\ `\ A \subseteq A \implies inj\_on\ f\ A \implies f\ `\ A = A.$$

For the maps we use we prove injectivity and show surjectiveity using this.

Lemma 2 (Transformations on uniform distributions).

1. $map_{spmf}\ (\lambda b.\ (y - b)\ mod\ q)\ (sample\_uniform\ q) = sample\_uniform\ q.$

2. $map_{spmf}\ (\lambda b.\ (y + b)\ mod\ q)\ (sample\_uniform\ q) = sample\_uniform\ q$.
3. $map_{spmf}\ (\lambda b.\ (y + x.b)\ mod\ q)\ (sample\_uniform\ q) = sample\_uniform\ q$.

Proof. These follow with the help of Lemma 1. Case 3 holds only under the additional assumption that x and q are coprime. This will always be the case in the applications we consider as $x \in \mathbb{Z}_q$ and q is a prime. □

3 Computational Indistinguishability in Isabelle

We introduce the notion of computational indistinguishability as the definitions of security we give in Section 4 rely on it. We use the definition from [14].

Definition 1. *A probability ensemble $X = \{X(a, n)\}$ is a sequence of random variables indexed by $a \in \{0,1\}^*$ and $n \in \mathbb{N}$. Two ensembles X and Y are said to be computationally indistinguishable, written $X \stackrel{c}{\equiv} Y$, if for every non-uniform polynomial-time algorithm D there exists a negligible function[3] ϵ such that for every a and every $n \in \mathbb{N}$,*

$$|Pr[D(X(a, n)) = 1] - Pr[D(Y(a, n)) = 1]| \leq \epsilon(n)$$

The original definition restricts $a \in \{0,1\}^*$, but we generalise this to an arbitrary first-order type, α. We model a probability ensemble as having some input of of this type, and a natural number security size parameter. The space of events considered depends on the *view*; also of arbitrary first-order type, ν.

$$type\_synonym\ (\alpha, \nu)\ ensemble = \alpha \Rightarrow nat \Rightarrow \nu\ spmf$$

We do not formalise a notion of polynomial-time programs in Isabelle as we do not need it to capture the following proofs. In principle this could be done with a deep embedding of a programming language, its semantic denotation function and a complexity measure. Instead, we will assume a family of constants giving us the set of all polynomial-time distinguishers for every type ν, indexed by a size parameter.

A polynomial-time distinguisher "characterises" an arbitrary spmf.

$$consts\ polydist\ ::\ nat \Rightarrow (\nu\ spmf \Rightarrow bool\ spmf)\ set$$

Now we can formalise Definition 1 directly as:

```
comp_indist :: (α, ν) ensemble ⇒ (α, ν) ensemble ⇒ bool
where comp_indist X Y ≡
  ∀(D :: ν spmf ⇒ bool spmf).
      ∃ (ϵ :: nat ⇒ real). negligible ϵ ∧
          (∀ (a :: α) (n :: nat).
              (D ∈ polydist n) ⟶
                  |spmf (D (X a n)) True − spmf (D (Y a n)) True| ≤ ϵ n))
```

[3] A negligible function is a function $\epsilon :: \mathbb{N} \to \mathbb{R}$ such that for all $c \in \mathbb{N}$ there exists $N_c \in \mathbb{N}$ such that for all $x > N_c$ we have $|\epsilon(x)| < \frac{1}{x^c}$

4 Semi-Honest Security and Simulation-Based Proofs

In this section we first define security in the semi-honest adversary model using the simulation-based approach. We then show how we use a probabilistic programming framework to formally prove security.

A protocol is an algorithm that describes the interaction between parties and can be modelled as a set of probabilistic programs. A two party protocol π computes a map from pairs of inputs to pairs of outputs. This map is called the protocol's *functionality* as it represents the specification of what the protocol should achieve. It can be formalised as a pair of (potentially probabilistic) functions

$$f_1 : input_1 \times input_2 \longrightarrow output_1$$

$$f_2 : input_1 \times input_2 \longrightarrow output_2$$

which represent each party's output independently. The composed pairing is the functionality, f, of type

$$f : input_1 \times input_2 \longrightarrow output_1 \times output_2$$

where $f = (f_1, f_2)$. That is, given inputs (x, y) the functionality outputs $(f_1(x, y), f_2(x, y))$. This indicates that party one gets $f_1(x, y)$ and party two gets $f_2(x, y)$ as output. In general the types of inputs and outputs can be arbitrary. For our instantiation we use concrete types depending on the functionality concerned.

For the initial example secure multiplication protocol we consider in Section 5 we have the probabilistic functionality $f(x, y) = (s_1, s_2)$ where $s_1 + s_2 = x.y$. Each party obtains an additive share of the multiplication. The protocol is run using a publicly known field $\mathbb{Z}_q$ where q is a prime number dependent on the security parameter. To ensure neither of the outputs alone reveal the value of $x.y$, we uniformly sample one of the outputs in the functionality

$$f(x, y) = (s_1, x.y - s_1), s_1 \xleftarrow{\$} \mathbb{Z}_q \tag{4}$$

The notation $s_1 \xleftarrow{\$} \mathbb{Z}_q$ means we sample s_1 uniformly from $\mathbb{Z}_q$. The Isabelle definition of the functionality is given below. It makes use of the do notation:

$$\begin{aligned} &f\ x\ y = do\ \{ \\ &\quad s_1 \leftarrow sample\_uniform\ q; \\ &\quad return_{spmf}(s_1, x.y - s_1)\} \end{aligned}$$

This functionality is easy to compute if one does not consider security; the parties can share their inputs and compute it. But with the security requirement that neither party learns anything about the others' input the problem becomes harder. We will give a protocol that securely computes this functionality later. We first introduce the notions used to define security. Security is based on *views* which capture the information known by each party. We follow the definitions given by Lindell in [14] to define security in the semi-honest model.

Definition 2. *Let π be a two party protocol with inputs (x, y) and with security parameter n.*

- *The real view of the i^{th} party (here $i \in \{1, 2\}$) is denoted by*

$$view_i^{\pi}(x, y, n) = (w, r^i, m_1^i, ..., m_t^i)$$

  where $w \in \{x, y\}$ and is dependent on which view we are considering, r^i accumulates random values generated by the party during the execution of the protocol, and the m_j^i are the messages received by the party.
- *Denote the output of the i^{th} party, $output_i^{\pi}(x, y, n)$, and the joint output as*

$$output^{\pi}(x, y, n) = (output_1^{\pi}(x, y, n), output_2^{\pi}(x, y, n)).$$

Definition 3. *A protocol π is said to securely compute f in the presence of a semi-honest adversary if there exist probabilistic polynomial time algorithms (simulators) S_1, S_2 such that*

$$\{S_1(1^n, x, f_1(x, y)), f(x, y)\} \stackrel{c}{\equiv} \{view_1^{\pi}(x, y, n), output^{\pi}(x, y, n)\}$$

$$\{S_2(1^n, y, f_2(x, y)), f(x, y)\} \stackrel{c}{\equiv} \{view_2^{\pi}(x, y, n), output^{\pi}(x, y, n)\}.$$

A semi-honest adversary is one that follows the protocol description. The simulator is given a unary encoding of the security parameter.

This definition formalises the idea that a protocol is secure if whatever can be computed by a party can also be computed from only the input and output of the party meaning that nothing extra is learned from the protocol.

For the secure multiplication protocol and the receiver's security in the Naor-Pinkas OT we prove security in an information theoretic sense. This means even computationally unbounded adversaries cannot gain extra information from the protocol. This is shown by proving the two sets of distributions above are equal. Information theoretic security is a stronger notion of security than computational indistinguishability and Isabelle proves the former implies the latter with ease.

A functionality is deterministic if given inputs always produce the same output. For a deterministic protocol it is shown in [14] that the above definition can be relaxed. We require correctness and

$$\{S_1(1^n, x, f_1(x, y))\} \stackrel{\text{c}}{\equiv} \{view_1^{\pi}(x, y, n)\} \tag{5}$$

$$\{S_2(1^n, y, f_2(x, y))\} \stackrel{\text{c}}{\equiv} \{view_2^{\pi}(x, y, n)\} \tag{6}$$

For a protocol to be correct we require that for all x, y and n there exists a negligible function μ such that

$$Pr[output^{\pi}(x, y, n) \neq f(x, y)] \leq \mu(n).$$

The Naor-Pinkas OT protocol, and the OT we use in the AND gate protocol given later, are both deterministic. The secure multiplication protocol however is not. For the deterministic cases we will focus on the more interesting property, showing the views are equal. As such when we refer to a deterministic protocol as being secure we explicitly show Equations 5 and 6 and assume correctness. For the non-deterministic secure multiplication protocol we must show exactly the property given in Definition 3.

4.1 Probabilistic Programming used for Simulation-Based Proofs

CryptHOL provides a strong foundation from which to manipulate and show equivalence between probabilistic programs. So far it has only been used to prove security in the game-based setting. The game-based definitions of security use a game played between an adversary and a benign challenger. The players are modelled as probabilistic programs and communicate with each other. The definition of security is tied to some event which is defined as the output of the security game. In general, proofs describe a reduction of a sequence of games (probabilistic programs) that end in a game where it can be shown the adversary has the same advantage of winning over the challenger as it would have against a problem assumed to be hard. The games in the sequence are then shown to be equivalent. This is shown on the left hand side of Fig. 1. We use a probabilistic

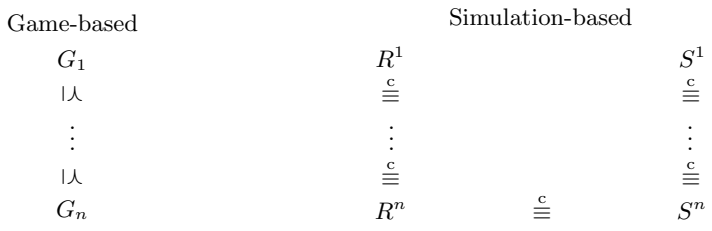

Fig. 1. A comparison between the game-based and simulation-based approaches. The game-based approach uses reductions (denoted $\preceq$) whereas in the simulation approach we show computational indistinguishability between probabilistic programs.

programming framework to construct simulation-based proofs. Our method of proof models the simulator and the real view of the protocol as probabilistic programs. In the right hand side of Fig. 1 we start with the real view of the protocol, R^1, and the simulator, S^1. We define a series of intermediate probabilistic programs (R^i, S^i) which we show to be computationally indistinguishable (or equal in the case of information theoretic security) — this is referred to as the *hybrid argument* in cryptography. This sequence ends in R^n and S^n which we show to be computationally indistinguishable (or equal). We have shown the diagram for the simulation-based approach in Fig. 1 is transitive.

Lemma 3. *Let X, Y and Z be probability ensembles then we have*

$$[X \overset{c}{\equiv} Y;\ Y \overset{c}{\equiv} Z] \implies X \overset{c}{\equiv} Z.$$

For the non-deterministic secure multiplication protocol we will construct the protocol and functionality outputs in the real and simulated views, instead of constructing them separately and combining them to form the ensembles.

5 Secure Multiplication Protocol

We now present a protocol that computes the functionality in Equation 4. The protocol requires some pre-generation of elements to be distributed to the parties. This is known in MPC as the preprocessing model [5], where the parties run an offline phase to generate correlated random triples — sometimes called Beaver triples — that are used to perform fast secure multiplications in an online phase. For this task we assume a trusted initialiser that aids in the computation. We

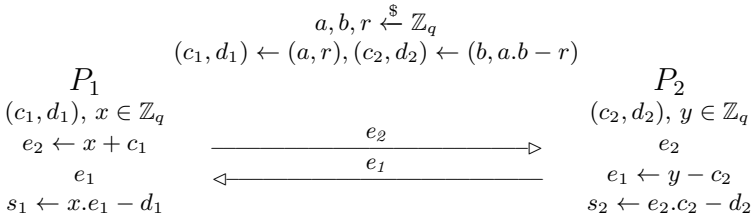

Fig. 2. A protocol for secure multiplication

denote the assignment of variables by $a \leftarrow b$ and all operations are taken modulo q. The claim of security is:

Theorem 1. *The protocol in Fig. 2 securely computes the functionality given in Equation 4 in the semi-honest adversary model.*

Intuitively, security results from the messages being sent in the protocol always being masked by some randomness. In the message party one sends, e_2, the input (x) is masked by the uniform sample, c_1. Likewise in the message party two sends, e_1, the input (y) is masked by the uniform sample, c_2.

5.1 Formal Proof of Security

The simulator and the real view of party one are defined in Isabelle as in Fig. 3. Recall that the protocol output is output by the real view and the functionality is output by the simulated view for this non-deterministic case. Thus we sample b and r twice (second time as b', r') in the real view. The outputs o_1 and o_2 refer to the output ($output^{\pi}(x, y, n)$ in Definition 3) of the protocol. Note that the simulator S_1 takes x and y as inputs. The simulated view however is constructed using only party one's input, x, according to the definition in Section 4. The second input, y, is used to construct the functionality output at the same time.

To show information theoretic security we prove that the two probabilistic programs given in Fig 3 are equal. This involves a series of small equality steps between intermediate probabilistic programs as shown in Fig 1. In particular, in the series of intermediate programs we manipulate the real and simulated views

$S_1\ x\ y = do\ \{$
$\quad a,\ b,\ c \leftarrow sample\_uniform\ q;$
$\quad s_1 \leftarrow sample\_uniform\ q;$
$\quad let\ z = (x.b - c)\ mod\ q;$
$\quad let\ s_2 = (x.y - s_1)\ mod\ q;$
$\quad return_{spmf}(x, a, z, b, s_1, s_2)\}$

$R_1\ x\ y = do\ \{$
$\quad a,\ b,\ r \leftarrow sample\_uniform\ q;$
$\quad b',\ r' \leftarrow sample\_uniform\ q;$
$\quad let\ z = (y - b)\ mod\ q;$
$\quad let\ o_1 = (x(y - b') - r')\ mod\ q;$
$\quad let\ o_2 = (x.y - (x(y - b') - r'))\ mod\ q;$
$\quad return_{spmf}(x, a, r, z, o_1, o_2)\}$

Fig. 3. Probabilistic programs to output the real and simulated views for party one.

into a form where we can apply Lemma 2(1). To do this we mainly use existing lemmas from CryptHOL, two of which are given in Equations 2 and 3.

This gives us the first half of formal security which can be seen in Lemma 4

Lemma 4. *For all inputs x and y we have, $S_1\ x\ y = R_1\ x\ y$. This implies the definition of security we gave in Sect. 4, $S_1\ x\ y \overset{c}{\equiv} R_1\ x\ y$.*

The proof of security for party two is analogous and together, Lemmas 4 and 5 establish Theorem 1.

Lemma 5. *For all inputs x and y we have, $S_2\ x\ y = R_2\ x\ y$. This implies the definition of security we gave in Sect. 4, $S_2\ x\ y \overset{c}{\equiv} R_2\ x\ y$.*

6 Naor-Pinkas Protocol

In the Naor-Pinkas OT protocol [19] we work with a cyclic group $\mathbb{G}$ of order q where q is a prime, for which the DDH assumption holds. The Decisional Diffie Hellman (DDH) assumption [10] is a computational hardness assumption on cyclic groups. Informally, the assumption states that given g^a and g^b, where a and b are uniform samples from $\mathbb{Z}_q$, the group element $g^{a.b}$ looks like a random element from $\mathbb{G}$. A triple of the form $(g^a, g^b, g^{a.b})$ is called a DDH triple. In the protocol, given in Fig 4, the Sender (party one) begins with input messages $(m_0, m_1) \in \mathbb{G}^2$ and the Receiver (party two) begins with $v \in \{0, 1\}$, the choice bit. At the end of the protocol the receiver will know m_v but will learn nothing about m_{1-v} and the sender will not learn v.

We prove information theoretic security in the semi-honest model for the receiver. Security for the sender is proven with a reduction to the DDH assumption. In particular, the receiver is only able to decrypt m_v as the corresponding ciphertext is a valid ElGamal ciphertext, while m_{1-v} is garbage.

In the protocol description, given in Fig 4, DDH-SR refers to a DDH *random self reduction* operation which takes DDH triples to DDH triples and non DDH triples to non DDH triples. The reduction is defined as follows. Given an input tuple (g, g^x, g^y, g^z), one picks a, b uniformly from $\mathbb{Z}_q$ and outputs $(g, g^{(x+b)a}, g^y, g^{(z+b.y)a})$. The role of the DDH random self reduction is to destroy any partial information in the message the Receiver sends to the Sender.

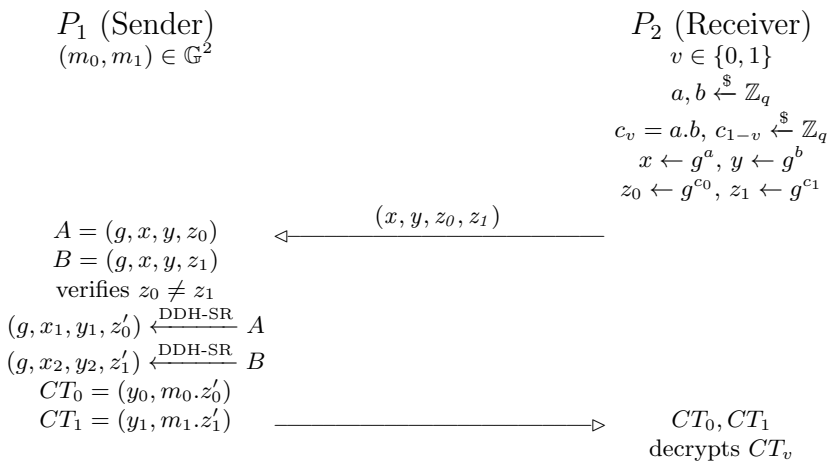

Fig. 4. The Naor-Pinkas OT protocol

Theorem 2. *The protocol defined in Fig. 4 securely computes a 1-out-of-2 OT in the semi-honest adversary model.*

6.1 The Formal Proof

We have a deterministic protocol and so do not include the overall functionality as part of the views. We must first consider the DDH-SR. In particular the two cases, when the input tuple is a DDH triple and when it is not. In both cases we simplify the operation that is performed. The simplified definitions are given in Fig 5 and the formal statements in Lemmas 6 and 7:

$$\begin{array}{l} DDH\_SR\_triple\ x\ y\ z = do\ \{ \\ \quad x_1 \leftarrow sample\_uniform\ q; \\ \quad return_{spmf}(g, g^{x_1}, g^y, g^{y.x_1 \ mod\ q})\} \end{array} \qquad \begin{array}{l} DDH\_SR\_non\_triple\ x\ y\ z = do\ \{ \\ \quad x_1, x_2 \leftarrow sample\_uniform\ q; \\ \quad return_{spmf}(g, g^{x_1}, g^y, g^{x_2})\} \end{array}$$

Fig. 5. The two simplified probabilistic programs for the DDH triples and non-triples.

Lemma 6. *For all x, y, z such that $z = y.x \bmod q$ we have*

$$DDH\_SR\ x\ y\ z = DDH\_SR\_triple\ x\ y\ z.$$

Lemma 7. *For all x, y, z such that $z \neq y.x \bmod q$ we have*

$$DDH\_SR\ x\ y\ z = DDH\_SR\_non\_triple\ x\ y\ z.$$

The Simulators and Views. First we consider party two. In constructing the real and simulated views we use the assert function to ensure the condition given in the protocol in Fig 4, $z_0 \neq z_1$, holds. This ensures that only one of A and B is a DDH triple; the other is not and hence the corresponding ciphertext CT_0 or CT_1 cannot be decrypted. The simulator may take as inputs $v \in \{0, 1\}$ and CT_v (although does not require it). We use $\otimes$ to denote multiplication in the group (as in Isabelle). The real view and simulator are shown below.

$S_2\ v = do\ \{$
$\quad a, b \leftarrow sample\_uniform\ q;$
$\quad let\ c_v = a.b;$
$\quad c'_v \leftarrow sample\_uniform\ q;$
$\quad \_ \leftarrow assert\_spmf(c'_v \neq b.a\ mod\ q);$
$\quad x_0 \leftarrow sample\_uniform\ q;$
$\quad x_1 \leftarrow sample\_uniform\ q;$
$\quad return_{spmf}(v, a, b, c'_v, g^b, g^{x_1}, g^b, g^{x_2})\}$

$R_2\ m_0\ m_1\ v = do\ \{$
$\quad a, b \leftarrow sample\_uniform\ q;$
$\quad let\ c_v = a.b;$
$\quad c'_v \leftarrow sample\_uniform\ q;$
$\quad \_ \leftarrow assert\_spmf(c'_v \neq b.a\ mod\ q);$
$\quad (g, x_0, y_0, z'_0) \leftarrow DDH\_SR\ a\ b\ c_v;$
$\quad (g, x_1, y_1, z'_1) \leftarrow DDH\_SR\ a\ b\ c'_v;$
$\quad let\ e_0 = z'_0 \otimes m_0;$
$\quad let\ e_1 = z'_1 \otimes m_1;$
$\quad return_{spmf}(v, a, b, c'_v, y_0, e_0, y_1, e_1)\}$

For party one, the simulator, S_1, takes in the two messages (m_0, m_1) (again, it does not use them) and the Sender's output - which amounts to nothing. We break this proof down into cases on v, the receivers input, however it is important to stress that the simulator must stay the same in both cases. Below we give the simulator and the real view for the non trivial case, namely when $v = 1$.

$S_1\ m_0\ m_1 = do\ \{$
$\quad a, b, c_1 \leftarrow sample\_uniform\ q;$
$\quad return_{spmf}(g^a, g^b, g^{a.b}, g^{c_1})\}$

$R_1\_v\_eq\_1\ m_0\ m_1 = do\ \{$
$\quad a, b, c_0 \leftarrow sample\_uniform\ q;$
$\quad return_{spmf}(g^a, g^b, g^{c_0}, g^{a.b})\}$

Proof of Security for the Receiver. From the construction of the real view one can see the triple (a, b, c_v) is a DDH triple and (a, b, c'_v) is not. Thus we are able to rewrite the real view using Lemmas 6 and 7.

The only components of the outputs of R_2 and S_2 which differ, up to unfolding of definitions are the encryptions. In the real view they are of the form $g^z \otimes m_i$ where z is uniformly sampled and in the simulator they are of the form g^z. We utilise a lemma from CryptHOL which states that if $c \in carrier\ \mathbb{G}$ then:

$$map_{spmf}\ (\lambda x.\ g^x \otimes c)\ (sample\_uniform\ q) = map_{spmf}\ (\lambda x.\ g^x)\ (sample\_uniform\ q)$$

This allows us to show our security result stated in Lemma 8.

Lemma 8. *For all inputs m_0, m_1 and v we have, $S_2\ v = R_2; m_0\ m_1\ v$. This implies the definition of security we gave in Sect. 4, $S_2\ v \overset{c}{\equiv} R_2; m_0\ m_1\ v$.*

Proof of security for the Sender. For $v = 0$, the proof is trivial as the simulator and real views are constructed in exactly the same way.

Lemma 9. *The case of $v = 0$ for party one implies for all inputs m_0 and m_1,*

$$R_1\_v\_eq\_0\ m_0\ m_1 = S_1\ m_0\ m_1.$$

The proof for $v = 1$ is equivalent to showing the distributions $(g^a, g^b, g^{a.b}, g^c)$ and $(g^a, g^b, g^c, g^{a.b})$ are computationally indistinguishable, when a, b, c are uniformly sampled. Here we provide a high level view of the pencil and paper. Our formalisation can be found at https://github.com/alan-turing-institute/isabelle-mpc.

To show security we provide a reduction to the DDH assumption, which implies the two distributions are computationally indistinguishable. In particular we show that if there exists a D that can distinguish the above two 4-tuples then one can construct an adversary that breaks the DDH assumption. In order to prove this formally we provide a way of formalising the DDH advantage.

Definition 4. *The DDH advantage for a distinguisher D is defined as*

$$\begin{aligned} adv\_ddh(D) = Pr[D((g^a, g^b, g^{a.b}), (g^a, g^b, g^c)) = 1] \\ - Pr[D((g^a, g^b, g^c), (g^a, g^b, g^{a.b})) = 1] \end{aligned}$$

where $a, b, c \xleftarrow{\$} \mathbb{Z}_q$.

We assume that no efficient distinguisher has an advantage greater than a negligible function of the security parameter. We define the advantage of a 4-tuple distinguisher, D, as follows.

Definition 5. *The 4-tuple distinguisher's advantage is given by*

$$\begin{aligned} adv\_dist(D) = Pr[D((g^a, g^b, g^{a.b}, g^c), (g^a, g^b, g^c, g^{a.b})) = 1] \\ - Pr[D((g^a, g^b, g^c, g^{a.b}), (g^a, g^b, g^{a.b}, g^c)) = 1] \end{aligned}$$

where $a, b, c, d \xleftarrow{\$} \mathbb{Z}_q$.

The adversary we use to break the DDH assumption, which uses D is constructed below.

DDH Adversary, inputs: $(g^a, g^b, g^{a.b})$ and (g^a, g^b, g^c).

- The adversary constructs $a_1 = (g^a, g^b, g^c, g^{a.b})$ and $a_2 = (g^a, g^b, g^{a.b}, g^c)$ and gives them to D.
- The adversary outputs whatever D outputs.

We show the DDH advantage of the adversary (using D) is the same as the 4-tuple advantage of D. Thus we have reduced the security of party one to a known hard problem. In particular we show

Lemma 10. *For any 4-tuple distinguisher D we have,*

$$adv\_ddh(A(D)) = adv\_dist(D).$$

This along with showing information theoretic security (Lemma 8) for the receiver means we have shown the protocol to be secure in the semi-honest model.

7 Towards Evaluating Arbitrary Functionalities

Several MPC techniques allow for the secure joint evaluation of *any* functionality represented as a Boolean circuit or an arithmetic circuit. At a high level, these protocols proceed by evaluating the circuit gate by gate while always keeping a secret share of the partial evaluation. In particular the GMW protocol relies on OT to securely evaluate AND gates

In this section we use a basic OT protocol (Fig 6) to construct a protocol to compute the output of an AND gate. The OT protocol we use employs a trusted initialiser, like the secure multiplication protocol of Section 5. The trusted initialiser pre-distributes correlated randomness to the parties so they can carry out the protocol. In particular r_0 and r_1 are uniformly sampled and given to party one, and d is uniformly sampled and given to party two along with r_d. The AND gate protocol then uses OT, this is done in a similar way as in the GMW protocol. The AND gate protocol we use here is taken from [6] and is described in Fig. 7. This demonstrates that OT can be used in powerful ways to construct protocols to compute fundamental functions securely.

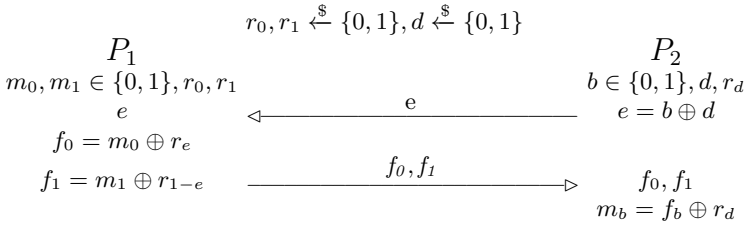

Fig. 6. Single bit OT

Initially we show information theoretic security for the OT construction given in Fig 6. That is we construct simulators S_1^{OT} and S_2^{OT} such that for the appropriately defined views R_1^{OT} and R_2^{OT} the result in Lemma 11 holds. To do this we define an appropriate XOR function ($\oplus$) on Booleans and prove a one time pad lemma on the XOR function.

Lemma 11. $R_1^{OT}\ m_0\ m_1\ b = S_1^{OT}\ m_0\ m_1$ *and* $R_2^{OT}\ m_0\ m_1\ b = S_2^{OT}\ b$.

We now define a protocol (Fig 7) to compute an AND gate. The protocol uses OT as a black box to transfer m_b. Each party outputs an additive share of the desired AND gate output. This protocol is proved secure using the simulation-based approach. We use Lemma 11 to prove security of this protocol in the semi-honest model. The real view and the simulator for party A are given in Fig

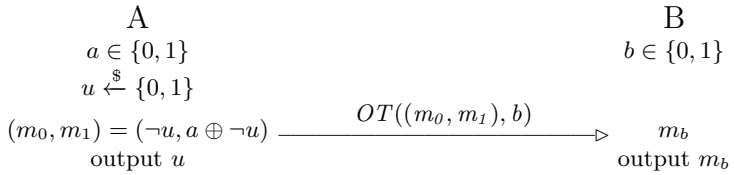

Fig. 7. A protocol to compute an AND gate

8. The simulator for party B, S_B, is constructed in an analogous way. Using these simulators we are able to show the AND gate protocol in Fig 7 is information theoretically secure.

$R_A\ a\ b = do\ \{$
$\quad u \leftarrow coin\_spmf;$
$\quad let\ m_0 = \neg u;$
$\quad let\ m_1 = a \oplus (\neg u);$
$\quad r \leftarrow R_1^{OT}\ m_0\ m_1\ b;$
$\quad return_{spmf}(u, r)\}$

$S_A\ a = do\ \{$
$\quad u \leftarrow coin\_spmf;$
$\quad let\ m_0 = \neg u;$
$\quad let\ m_1 = a \oplus (\neg u);$
$\quad r \leftarrow S_1^{OT}\ m_0\ m_1;$
$\quad return_{spmf}(u, r)\}$

Fig. 8. Simulator and real view of party A

Lemma 12. *Information theoretic security for the AND gate protocol is shown by the equalities*

$$R_A\ a\ b = S_A\ a\ and\ R_B\ a\ b = S_B\ b.$$

We have shown how a simple OT that uses a trusted initialiser can help to securely compute an AND gate. In general a trusted initialiser would not be necessary as one can use the N-P OT in the AND gate protocol. There is one technical issue with doing this. In the N-P OT we work with a group with multiplication but the AND gate protocol requires addition. In practice this is overcome by implementing the N-P OT using a ring (which has both operations), for which the DDH assumption holds. The proof would follow as in the proof given above, but an extension of the theory of rings in Isabelle is required for this - something we plan to develop in future work.

8 Conclusion

We have shown a general approach for capturing simulation-based cryptographic proofs in the computational model, building on Lochbihler's CryptHOL framework, and giving a proof of the Naor-Pinkas OT protocol. We also have shown how out technique can be used to formally prove security of a simple two party protocol for an AND gate based on OT.

Future Work. The work presented here is only a starting point for the development of theory and examples of simulation-based proofs. Oblivious Transfer is a fundamental cryptographic primitive which can be used to construct generic protocols for MPC. For example, Yao's garbled circuits use OT as a sub-protocol to exchange garbled inputs, while the GMW protocol relies on OT for computing AND gates. Section 7 took a first step towards a formal proof of the GMW protocol. Section 7 took a first step towards a formal proof of the GMW protocol. We plan to extend this work towards formalising general MPC protocols.

Related Work. Many formal techniques and tools have been devised which use the symbolic model. Work on formalising proofs in the computational model has begun more recently and is more challenging, requiring mathematical reasoning about probabilities and polynomial functions, besides logic. The CertiCrypt [2] tool built in Coq helped to capture the reasoning principles that were implemented directly in the dedicated interactive EasyCrypt tool [3]. Again in Coq, the Fundamental Cryptographic Framework [20] provides a definitional language for probabilistic programs, a theory that is used to reason about programs, and a library of tactics for game-based proofs. Interactive tools seem invaluable for complex protocols or exploring new techniques, but automatic tools are more practical when things become routine. CryptoVerif [7] is a tool with a high level of automation but its scope only stretches to secrecy and authentication in protocols. AutoG&P [4] is another automated tool dedicated to security proofs for pairing-based cryptographic primitives. So far, all of these tools have been used to perform game-based cryptographic proofs and not simulation-based proofs.

Acknowledgements. We are deeply grateful to Andreas Lochbihler for providing and continuing to develop CryptHOL and for his kind help given with using it. Also we are thankful to the reviewers for their comments regarding the presentation of our work.

References

1. M Abadi and P Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *J. Cryptology*, 20(3):395, 2007.
2. G Barthe, B Grégoire, and S Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *POPL*, pages 90–101. ACM, 2009.
3. G Barthe, B Grégoire, S Heraud, and S Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90. Springer, 2011.

4. G Barthe, B Grégoire, and B Schmidt. Automated proofs of pairing-based cryptography. In *ACM Conference on Computer and Communications Security*, pages 1156–1168. ACM, 2015.
5. D Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991.
6. C Bennett, G Brassard, C Crépeau, and M Skubiszewska. Practical quantum oblivious transfer. In *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 351–366. Springer, 1991.
7. B Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Trans. Dependable Sec. Comput.*, 5(4):193–207, 2008.
8. D Bogdanov, S Laur, and J Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS*, pages 192–206. Springer, 2008.
9. D Demmler, T Schneider, and M Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *NDSS*. The Internet Society, 2015.
10. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.
11. D Dolev and A Yao. On the security of public key protocols. *IEEE Trans. Information Theory*, 29(2):198–207, 1983.
12. O Goldreich, S Micali, and A Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.
13. M Keller, E Orsini, and P Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *ACM Conference on Computer and Communications Security*, pages 830–842. ACM, 2016.
14. Y Lindell. How to simulate it - A tutorial on the simulation proof technique. *IACR Cryptology ePrint Archive*, 2016:46, 2016.
15. Y Lindell and B Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.
16. Y Lindell, B Pinkas, N P. Smart, and A Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In *CRYPTO (2)*, volume 9216 of *Lecture Notes in Computer Science*, pages 319–338. Springer, 2015.
17. C Liu, X Shaun Wang, K Nayak, Y Huang, and E Shi. ObliVM: A programming framework for secure computation. In *IEEE Symposium on Security and Privacy*, pages 359–376. IEEE Computer Society, 2015.
18. A Lochbihler. Probabilistic functions and cryptographic oracles in higher order logic. In *ESOP*, volume 9632 of *Lecture Notes in Computer Science*, pages 503–531. Springer, 2016.
19. M Naor and B Pinkas. Efficient oblivious transfer protocols. In *SODA*, pages 448–457. ACM/SIAM, 2001.
20. A Petcher and G Morrisett. The foundational cryptography framework. In *POST*, volume 9036 of *Lecture Notes in Computer Science*, pages 53–72. Springer, 2015.
21. V Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive*, 2004:332, 2004.
22. A Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167. IEEE Computer Society, 1986.
23. S Zahur and D Evans. Obliv-C: A language for extensible data-oblivious computation. *IACR Cryptology ePrint Archive*, 2015:1153, 2015.