On the Formalisation of Σ-Protocols and Commitment Schemes

David Butler[1,2](✉), David Aspinall[1,2], and Adrià Gascón[1,3]

[1] The Alan Turing Institute, London, UK
dbutler@turing.ac.uk
[2] University of Edinburgh, Edinburgh, UK
[3] University of Warwick, Coventry, UK

Abstract. There is a fundamental relationship between Σ-protocols and commitment schemes whereby the former can be used to construct the latter. In this work we provide the first formal analysis in a proof assistant of such a relationship and in doing so formalise Σ-protocols and commitment schemes and provide proofs of security for well known instantiations of both primitives.

Every definition and every theorem presented in this paper has been checked mechanically by the Isabelle/HOL proof assistant.

Keywords: Commitment schemes · Σ-protocols · Formal verification · Isabelle/HOL

1 Introduction

In [8], Damgard elegantly showed how Σ-protocols can be used to construct commitment schemes that are perfectly hiding and computationally binding and thus showed how these two fundamental cryptographic primitives are linked. The properties of the resulting commitment scheme rely on the security of the underlying Σ-protocol. The relationship between the two is natural as Σ-protocols can be considered the building block for zero knowledge, and it is well known that commitment schemes and zero knowledge protocols are strongly related [9].

When properties of fundamental primitives are linked in such a way it is interesting to study them formally using a proof assistant to more deeply understand the nature of the relationship. In fact the proof provided of the security of the construction of commitment schemes from Σ-protocols in [8] is brief; thus to study it formally one has to consider the properties in more detail.

To achieve such a goal one must first formalise both primitives and then show the desired relations between them with respect to the individual formalisations for either primitive. To formalise and instantiate a primitive using a proof assistant one must first formalise the security definitions, then define the protocol

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

that realises the primitive and then provide proofs in the proof assistant that show the defined security properties are met by the protocol.

As well as providing a deeper insight and more rigorous proof for properties in cryptography, formalisations also provide an increased confidence in cryptographic proofs. This increased level of rigour was called for by Halevi in 2005 [12], where it was proposed to approach the problem formally. One aspect of this approach is that security definitions are formally defined in an abstract way and then instantiated for different protocols that hold those security properties. The advantage of the abstract definitions is a human checker only needs to check these definitions for consistency with the literature to have confidence in the whole collection of proof. This is exactly the goal of this work.

In this paper, motivated by understanding the connection between Σ-protocols and commitment schemes we use the proof assistant Isabelle/HOL to formally reason about the two fundamental primitives and then show how a Σ-protocol can be used to construct a commitment scheme. Specifically we formally prove, with respect to our abstract definitions of commitment schemes, how the Schnorr Σ-protocol can be used to construct a perfectly hiding and computationally binding commitment scheme. In the process of achieving this we prove various Σ-protocols and commitment schemes secure in their own right.

To the best of our knowledge this is the first time the connection between Σ-protocols and commitment schemes has been considered using a proof assistant. Σ-protocols were considered in [5] and [2] and the Pedersen commitment scheme has been considered formally using EasyCrypt in [17]. We leave a discussion of the comparison of Isabelle/CryptHOL and EasyCrypt to Sect. 10.

Our formal proofs can be found at [1].

Contributions

- We provide abstract frameworks from which to reason about commitment schemes and Σ-protocols. We formalise this in Isabelle/HOL, but the structure could be used in other proof assistants.
- We instantiate this abstract framework to provide proofs of security for both primitives; the Pedersen commitment scheme, the Schnorr Σ-protocol and a second Σ-protocol based on the equality of discrete logs assumption.
- We use both the abstract frameworks to formally show how a commitment scheme can be constructed using the Schnorr Σ-protocol (following the work of [9]) and prove it secure with respect to our commitment scheme framework. In doing so we formally demonstrate the relationship between Σ-protocols and commitment schemes.
- To complete the proofs described, two other contributions were made. First, we had to define the discrete logarithm assumption in Isabelle; a notion that can be used by others in future proofs. Second, the adversary used to break this assumption, for a contradiction, outputs the division of two elements in a field. For technical reasons this is non trivial to do in Isabelle therefore we were required to develop a method to do this. This method, and its associated

proofs, can now be used by others completing cryptographic proofs inside Isabelle.
– All our protocols are shown secure in both the concrete and asymptotic cases.

Outline. In Sect. 2 we outline the structure of our formalisation and in Sect. 3 introduce the relevant theory of Isabelle/HOL and the main parts of CryptHOL [15]. In Sects. 4 and 6 we introduce our formalisation of Σ-protocols and commitment schemes receptively. Sections 5 and 7 show how we instantiate these abstract frameworks for the Schnorr and Pedersen protocols. We show how we link the two in Sect. 8 and how we construct a commitment scheme using the Schnorr Σ-protocol. Finally we conclude and discuss related work and provide a comparison with EasyCrypt in Sects. 9 and 10.

2 Formalisation Overview

In this section we first outline the structure of our formalisation and then discuss the process of instantiating the abstract frameworks to achieve formal proof in Isabelle. Then we discuss the proof method for the asymptotic security setting.

2.1 Outline of Formalisation

We begin our formalisation by abstractly defining the security properties required for both commitment schemes and Σ-protocols. This part of the formalisation is defined over abstract types, giving the flexibility for it to be instantiated for any protocol; this allows us to have confidence in the proof's integrity when considering a range of different protocols. The abstract nature of the formalisation will also allow others to use the definitions of security and structure we provide to prove security of other commitment schemes and Σ-protocols. Another benefit is we can prove some technical lemmas at the abstract level and have them at our disposal in any instantiation, thus reducing the workload for future proofs. A final advantage of the abstract definitions is that a human checker only needs to verify these definitions for consistency with the literature to have confidence in the whole collection of proof.

We instantiate the abstract frameworks to prove security of the Pedersen commitment scheme, the Schnorr Σ-protocol and a second Σ-protocol that uses a relation for the equality of discrete logarithms. Finally we use the algorithms that define the Schnorr protocol to construct a commitment scheme (as shown in [8]) and prove it secure with respect to our commitment scheme definitions using the properties obtained from the Σ-protocol proofs.

The work flow of this paper can be seen in Fig. 1 where an arrow implies the use of one theory (a formalised file in Isabelle) to achieve the next. For example we prove the 'Schnorr commitment' secure with respect to the 'Abstract Commitments' definitions and using the algorithms and properties of the 'Schnorr' protocol.

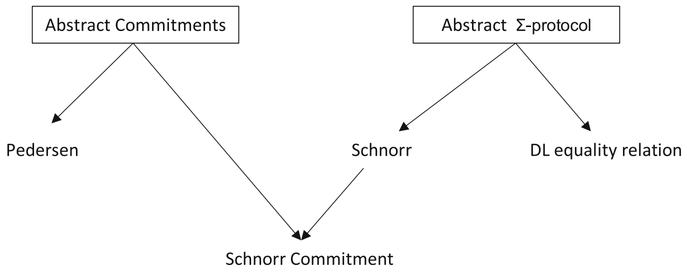

Fig. 1. The structure of the formalisation in Isabelle

2.2 Instantiating the Abstract Frameworks

At a technical level Isabelle's module system (called locales) allows the user to prove theorems abstractly, relative to given assumptions. These theorems can be reused in situations where the assumptions themselves are theorems. In our case locales allow us to define properties of security relative to fixed constants and then instantiate these definitions for explicit protocols and prove the security properties as theorems.

The overall process of instantiation can be seen as a step-by-step process given below:

1. We fix, with abstract types, the probabilistic programs (algorithms) that make up a primitive (e.g. *key_gen*, *commit*, *verify*, for commitment schemes see Fig. 6) in a locale then proceed to make the definitions of security with respect these fixed constants (e.g. define the hiding property). This can be considered as the formal specification requirements of the primitive.
2. To instantiate a protocol we must explicitly define the above fixed constants. To do this we make all types concrete and define the probabilistic programs that describe the protocol.
3. We are then able to utilise Isabelle's locale structure by importing the abstract framework using the **sublocale** command. Not only must the explicit definitions be of the correct type when importing a locale, one must also dismiss any assumptions that come with the locale. This means that our instantiation is a valid commitment scheme or Σ-protocol and allows us to refer to the security definitions made in the abstract framework and prove the properties using the explicit definitions of the instantiated protocol.

2.3 Asymptotic Security

In our formalisation we first consider security in the concrete setting. Here we assume the security parameter, n, is implicit in all algorithms that parametrise the framework. We then move to prove security in the asymptotic setting utilising Isabelle's module system. More details about this part of our formalisation are given in Sect. 8.1. We note the asymptotic setting is not considered in EasyCrypt proofs, the machinery available in Isabelle however makes it possible.

3 CryptHOL and Isabelle Background

In this section we briefly introduce the Isabelle notion we use throughout and then highlight and discuss some important aspects of CryptHOL. For more detail on CryptHOL see [6]. The full formalisation is available at [14].

3.1 Isabelle Notation

The notations we use in this paper resemble closely the syntax of Isabelle/HOL (Isabelle). For function application we write $f(x, y)$ in an uncurried form for ease of reading instead of $f\ x\ y$ as in the λ-calculus. To indicate that term t has type τ we write $t :: \tau$. Isabelle uses the symbol $\Rightarrow$ for the function type, so $a \Rightarrow b$ is the type of functions that takes an input of type a and outputs an element of type b. The type $'a$ denotes an abstract type. The implication arrow $\longrightarrow$ is used to separate assumptions from conclusions inside a closed HOL statement. We use $\otimes$ to denote multiplication in a group and $*$ for multiplication in a field.

3.2 CryptHOL

CryptHOL [6] is a framework for reasoning about cryptography in the computational model that is embedded inside the Isabelle/HOL theorem prover. It allows the prover to write probabilistic programs and reason about them. The computational model is based on probability theory and in particular uses probabilistic programs to define security—this can be seen for the construction of games in the game-based setting or the real and ideal views in the simulation-based setting.

To build the probabilistic programming framework CryptHOL uses the existing probability theory formalised inside Isabelle to define discrete probability distributions called sub probability mass functions (of type *spmf*). These can be thought of as probability mass functions with the property they do not have to sum to one—we can lose some probability mass. This allows us to model failure events and assertions.

Writing Probabilistic Programs. CryptHOL provides some, easy-to-read, Haskell-style do notation to write probabilistic programs where $\mathbf{do}\{x \leftarrow p;\ f(x)\}$ is the probabilistic program that samples from the distribution p and returns the *spmf* produced by f. We can also return an *spmf* using the monad operation *return*. See Fig. 2 for an example.

Proofs of security are mainly completed by manipulating the appropriate probabilistic programs. While the proofs that each manipulation is valid are not always accessible to non-experts, the effect of each manipulation can be easily seen and recognised as they are explicitly written in the do notation.

Failure Events and Assertions. We often have to reason about failure events. For example we must ensure the adversary in the hiding game (Fig. 6) outputs two valid messages for the game to proceed. Such events are handled using assertion statements

$$assert(b) = if\ b\ then\ return(\_)\ else\ \bot$$

and the $TRY\ p\ ELSE\ q$ construct. For example TRY **do** $\{p\}$ $ELSE\ q$ would distribute the probability mass not assigned by p to the distribution according to q. Picking up on our example of the hiding game; if the adversary fails to output two valid messages, the assertion fails and the $ELSE$ branch is invoked resulting in the adversary not winning the hiding game.

Sampling. Sampling from sets is important in cryptography. CryptHOL gives an operation *uniform* which returns a uniform distribution over a finite set. We use two cases of this function extensively: by $sample\_uniform(q)$, where q is a natural, we denote the uniform sampling from the set $\{..<\ q\}$ and by *coin* we denote the uniform sampling from the set $\{True, False\}$—a coin flip.

Using sampling we are able to illustrate one difference in thought process and rigour required in a formal proof compared to a pen-and-paper proof. One time pads (OTPs) are used extensively in protocols. Often their properties are employed without thought or explanation in paper proofs as they are considered to be a simple construct. However there are some more subtle issues that sometimes need to be considered.

$$map((\lambda b.\ (x * b)\ mod\ q), (sample\_uniform(q))) = sample\_uniform(q) \qquad (1)$$

Equation 1 shows the traditional OTP for multiplication in a field; a uniform sample, b, from a set of q elements, multiplied to an input, x, taken modulo q is the same as a uniform sample. However this property is only valid if x and q are coprime. This follows, in the finite field, from Fermat's Little Theorem; thus formally we have to work much harder to use such a lemma. In short, formalising a proof demonstrates many areas where a paper proof falls short in detail.

Probabilities. We must also be able to reason about the probability of events occurring. So, $\mathcal{P}[Q = x]$ denotes the subprobability mass the spmf Q assigns to the event x. We also introduce the notation $\triangleright$ which denotes the binding of a sample without the need for the do notation. This can be seen in Theorem 3 where the bound variable e is sampled from *challenge* and given to $S2$.

Negligible Functions. To reason about security in the asymptotic case we must consider negligible functions. These were formalised as a part of CryptHOL. A function, $f :: (nat \Rightarrow real)$ is said to be negligible if

$$(\forall c > 0.\ f \in o(\lambda x.inverse(x^c)))$$

where o is the little o notation. We discuss the use of such functions in our proofs in Sect. 8.1.

4 Formalising Σ-Protocols

In this section we show how we formally define Σ-protocols and their security properties—it is with respect to these definitions that we prove security of the Schnorr protocol and a variant of it for equality of discrete logs in Sect. 5. For more details on the Σ-protocols see [9].

4.1 Definition of Σ-protocols

We first consider a binary relation R; for some (h, w) that satisfies R, w is a witness of h. For example, the discrete log relation is formalised as follows

$$R_{DL}(h, w) = (h = g^w \wedge w < |G|) \tag{2}$$

where g is a generator of the cyclic group G.

A Σ-protocol is a two party protocol run between a prover (P) and a verifier (V). In the protocol h is a common input to both P and V and w a private input to P such that $R(h, w)$ is true. We define the structure of a Σ-protocol as follows:

Definition 1. *A Σ-protocol has the following three part form:*

1. *Initial message: P sends message a, created with randomness r.*
2. *Challenge: V sends P a challenge, e.*
3. *Response: P responds with z to convince the verifier who either accepts or rejects.*

A conversation *can be seen as the tuple (a, e, z).*

Formally we model this as four abstract probabilistic programs whose types are given below. The inputs to relation R are h, of type *'pub_input* and w, of type *'witness*.

$$\mathit{init} :: \text{'}\mathit{pub\_input} \Rightarrow \text{'}\mathit{witness} \Rightarrow (\text{'}\mathit{rand} \times \text{'}\mathit{msg})\ \mathit{spmf} \tag{3}$$

$$\mathit{challenge} :: \text{'}\mathit{challenge}\ \mathit{spmf} \tag{4}$$

$$\mathit{response} :: \text{'}\mathit{rand} \Rightarrow \text{'}\mathit{witness} \Rightarrow \text{'}\mathit{challenge} \Rightarrow \text{'}\mathit{response}\ \mathit{spmf} \tag{5}$$

$$\mathit{check} :: \text{'}\mathit{pub\_input} \Rightarrow \text{'}\mathit{msg} \Rightarrow \text{'}\mathit{challenge} \Rightarrow \text{'}\mathit{response} \Rightarrow \mathit{bool}\ \mathit{spmf} \tag{6}$$

The challenge sent by V is defined as a random sampling (see [9]) therefore it needs no inputs here.

It is interesting to note, unlike many paper based definitions, none of our algorithms in the formalisation need take random coins as input. This is because they are already probabilistic programs and thus not deterministic by definition.

The three properties that define a Σ-protocol are completeness, special soundness and honest verifier zero-knowledge (HVZK). Special soundness ensures the prover cannot prove a false statement and HVZK says the verifier learns nothing of the witness that it cannot learn from the output of the verification and the public input.

Definition 2. *Assume the protocol run between P and V has the above form then it is said to be a Σ-protocol if the following properties hold:*

- *Completeness: if P and V follow the protocol on public input h and private input w such that $R(h, w)$ is satisfied, then V always accepts.*

$$complete(h, w) \equiv R(h, w) \longrightarrow \mathcal{P}[complete\_game(h, w) = True] = 1$$

- *Special soundness: there exists an adversary, $\mathcal{A}$ such that when given a pair of accepting conversations (on public input h) (a, e, z) and (a, e', z') where $e \neq e'$ it can compute w such that $R(h, w)$ is satisfied.*

$$s\_soundness(h, w) \equiv$$
$$\exists \mathcal{A}.\ R(h, w) \longrightarrow \mathcal{P}[s\_soundness\_game(h, w, \mathcal{A}) = True] = 1$$

- *Honest verifier Zero-Knowledge: There exists a simulator S that on input h and challenge e outputs an accepting conversation (a, e, z) with the same probability distribution as the real conversations ($real\_view$) between P and V on input (h, w).*

$$HVZK(h, w) \equiv R(h, w) \longrightarrow (real\_view(h, w) = challenge \rhd (\lambda e.S(h, e)))$$

In the literature the adversary for the special soundness definition and the simulator in the HVZK definition must run in polynomial time. There are challenges in formalising this notion, therefore we visually verify that the adversaries we construct for special soundness run in polynomial time and do not provide a formalisation of this property.

We define the probabilistic program *complete_game* to run the components of the protocol in an honest way. In particular we define a probabilistic program that takes as input (h, w), and then runs the four probabilistic programs of the protocol as would be done in the protocol, finally outputting the output of *check*.

The probabilistic program *s_soundness_game* is slightly more subtle. The game takes as input $(h, w, \mathcal{A})$ and must construct two accepting views to give to the adversary. The condition on these views is that the challenge in the second view must be different to that of the first. On paper this is easy to reason about as it can be considered to be intuitive but formally we must work harder. We define a new probabilistic program, $snd\_challenge(e)$, that outputs a challenge different from the original. For example, for the Schnorr protocol the challenge is a uniform sample from the field. Consequently the second challenge must uniformly sample from all elements of the field modulo the first challenge p,

$$snd\_challenge(q, p) = uniform(\{.. < q\} - \{p\}) \tag{7}$$

We must then prove all properties we required of *challenge* with respect to the new definition.

In the honest verifier zero knowledge property the real view is a probabilistic program that defines the real view (i.e., the protocol) transcript of the execution, that is (a, e, z). Intuitively if one can simulate the real view then we know there is no leakage of data, in this case the witness, during an execution of the protocol. We note that unlike previous work on the simulation based proof method [7] in MPC where the real view could only be defined in the instantiation due to different protocol structures, here we can define it solely from the algorithms used in the Σ-protocol. Both the special soundness game and the definition of the real view can be seen in Fig. 2.

Having made the above definitions we can define Σ-protocols formally as follows.

Definition 3

$$\Sigma\text{-}protocol(h, w) = complete(h, w) \wedge s\_soundness(h, w) \wedge HVZK(h, w)$$

Referring back to the diagram in Fig. 1 we can see we have completed the work for the 'Abstract Σ-protocol' box.

```
$special\_soundness\_game(h, w, \mathcal{A}) = do\ \{$
    $(r, a) \leftarrow init(h, w);$
    $e \leftarrow challenge;$
    $z \leftarrow response(r, w, e);$
    $e' \leftarrow snd\_challenge(e);$
    $z' \leftarrow response(r, w, e');$
    $w' \leftarrow \mathcal{A}(h, (a, e, z), (a, e', z'));$
    $return(w = w')\}$
```

```
$real\_view(h, w) = do\ \{$
    $(r, a) \leftarrow init(h, w);$
    $e \leftarrow challenge;$
    $z \leftarrow response(r, w, e);$
    $return(a, e, z)\}$
```

Fig. 2. Definitions of the special soundness game and the real view for Σ-protocols.

5 Formalising the Schnorr Σ-Protocol

In this section we describe the proof of security of the Schnorr Σ-protocol. We also formalise the proof of security for a second Σ-protocol that is based on the equality of discrete logs [9]. The relation for this protocol can be seen in Eq. 8, where g' is a second generator of the cyclic group G.

$$R((h, h'), w) = \{h = g^w \wedge h' = g'^w \wedge w < |G|\} \tag{8}$$

In the interest of space here we only detail the formalisation of the Schnorr protocol. Here we provide more details of the formalisation than in other parts of the paper as well as a higher level commentary.

5.1 The Schnorr Σ-protocol

The Schnorr protocol uses a cyclic group G with generator g and considers the discrete log relation, R_{DL}, that can be seen in Eq. 2. The protocol is given in Fig. 3. The notation $\xleftarrow{\$}$ denotes uniform sampling while we use $\leftarrow$ to denote assignment.

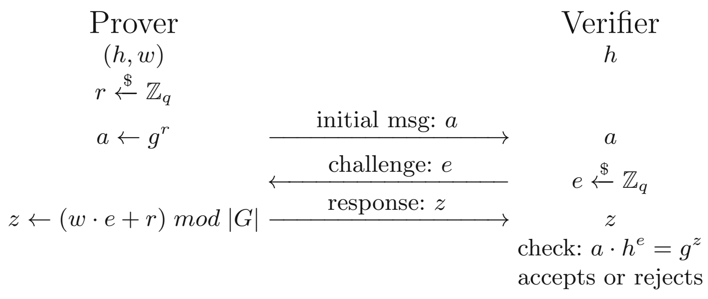
Prover
(h, w)
$r \xleftarrow{\$} \mathbb{Z}_q$
$a \leftarrow g^r$
$z \leftarrow (w \cdot e + r)\ mod\ |G|$
initial msg: a
challenge: e
response: z
Verifier
h
a
$e \xleftarrow{\$} \mathbb{Z}_q$
z
check: $a \cdot h^e = g^z$
accepts or rejects

Fig. 3. The Schnorr Σ-protocol.

We consider the three properties that define a Σ-protocol in turn.

Completeness comes directly by unfolding the definitions and proving the identity $g^r \otimes (g^w)^e = g^{r+w*e}$. This is trivial, but provides Isabelle with a hint as to how to rewrite the definitions to dismiss the completeness proof.

Theorem 1. *Assume $R_{DL}(h, w)$ then*

$$\mathcal{P}[completeness\_game(h, w) = True] = 1$$

Secondly we must prove special soundness. To prove this we must construct an adversary that can extract the witness from two correct executions of the protocol. The special soundness adversary is given in Fig. 4.

```
$\mathcal{A}_{ss}(h, c_1, c_2) = do\ \{$
    $let\ (a, e, z) = c1;$
    $let\ (a', e', z') = c_2;$
    $return(if\ e > e'\ then\ (z - z') * (fst(bezw((e - e'), |G|))\ mod\ |G|)$
                $else\ (z' - z) * (fst(bezw((e' - e), |G|))\ mod\ |G|))\}$
```

Fig. 4. The adversary used to prove special soundness for the Schnorr Σ-protocol. Note the output is equivalent to $\frac{z-z'}{e-e'}$. In the proof of Theorem 2 we have $a = a'$ for the messages given to the adversary.

We highlight an important contribution of our work here. The output of $\mathcal{A}_{ss}$ appears complex but actually is equivalent to $\mathcal{A}_{ss}$ outputting $\frac{z-z'}{e-e'}$ in the field. The output uses Bezout's function (*bezw*) for finding a pair of witnesses for Bezout's theorem to realise the inverse of $e - e'$. This function is given in the Isabelle number theory library.

The reason we could not define the adversary as outputting a simple division is worthy of some technical discussion. The inputs to the division are of type natural and thus any output from the division is also required to be of the same type as we are working in a field. However the output type of a division on naturals in Isabelle is a real number. Thus we must work around to output the correct value as a natural. The condition on $e > e'$ is such that the denominator is never negative as we are working with naturals. While this may look like an unnatural solution to the issue it is an effective one, and the work we provide here can be used by others when this problem arises again (it almost certainly will as division in a field is not uncommon in cryptography!). To allow us to work with an adversary defined in such a way we must prove lemmas of the form:

Lemma 1. *Assume* $a, b, w, y < |G|$, $a \neq b$ *and* $w \neq 0$ *then*

$$\begin{aligned} w = (\mathit{if}\ (a > b)\ \mathit{then}((w * a + y) - (w * b + y)) * \mathit{fst}(\mathit{bezw}((a - b), |G|) \\ \mathit{else}((w * b + y) - (w * a + y)) * \mathit{fst}(\mathit{bezw}((b - a), |G|) \end{aligned} \tag{9}$$

The proof of Lemma 1 is quite involved however lemmas we require for other instances follow a similar proof method. We also prove a general lemma to compute divisions in a finite field, this is given in Lemma 2.

Lemma 2. *Assume* $\mathit{gcd}(a, |G|) = 1$ *then*

$$[a * \mathit{fst}(\mathit{bezw}(a, |G|)) = \mathit{1}](\mathit{mod}|G|)$$

To apply statements such as Lemma 1 we often have to employ congruence rules. These allow the simplifier to use the context, in particular here on facts pertaining to the bound variables in the probabilistic programs that are required as assumptions. Using these methods we can prove special soundness.

Theorem 2. *Assume* $R(h, w)$ *then we have*

$$\mathcal{P}[\mathit{specical\_soundness\_game}(h, w, \mathcal{A}_{ss}) = \mathit{True}] = 1$$

Finally we must prove honest verifier zero knowledge. This requires us to define the real view of the protocol and show that there exists a simulator that takes as input the public input and a challenge and outputs a view that is indistinguishable from (equal as probability distributions) the real view. This technique follows the technique of simulation based proofs that was formally introduced in Isabelle in [7]. The probabilistic program defining the simulator along with the unfolded definition of the real view is given in Fig. 5.

To show HVZK we prove the two views are equal. That is,

Theorem 3. *Assume* $R_{DL}(h, w)$ *then we have*

$$\mathit{real\_view}(h, w) = (\mathit{challenge} \triangleright (\lambda e.\ S(h, e)))$$

In the definitions given in Fig. 5 the number of random samples is different in each view. We note that the extra sampling for the simulation comes from the challenge which, by definition is sampled before being given to the simulator. To prove honest verifier zero knowledge we manipulate the real view into a form where we can use Eq. 10, that describes a one time pad for addition in the field.

$$\mathit{map}(\lambda b.\ (y + b)\ \mathit{mod}\ q,\ \mathit{uniform}(q)) = \mathit{uniform}(q) \tag{10}$$

```
$real\_view(h, w) = do$ {                    $S(h, e) = do$ {
  $r \leftarrow uniform(|G|)$;                  $c \leftarrow uniform(|G|)$;
  $let(r, a) = (r, g^r)$;                       $let\ a = g^c \otimes (h^e)^{-1}$;
  $c \leftarrow uniform(|G|)$;                  $return\ (a, e, z)$}
  $let\ z = (w * c + r)\ mod\ |G|$;
  $return\ (a, c, z)$}
```

Fig. 5. The unfolded real view and simulator for the Schnorr protocol

To do this we must prove some basic identities about groups that provide Isabelle with hints as to rewrite the probabilistic programs. After proving the three properties we can show that the Schnorr protocol satisfies the definition of a Σ-protocol.

Theorem 4. *For the Schnorr Σ-protocol we have*

$$\Sigma\text{-}protocol(h, w).$$

6 Formalising Commitment Schemes

In this section we introduce our formalisation of commitment schemes and their security properties. Commitment schemes are a cryptographic primitive, run between a committer C and a verifier V, that allow the committer to commit to a chosen message, while keeping it private, and at a later time reveal the message that was committed to. For more details on commitment schemes we refer the reader to [20].

There are three phases to a commitment scheme:

1. Key generation, The key generation algorithm generates keys and sends them to both P and V respectively.
2. Commitment phase, The committer sends the verifier its committed value (or commitment), c, for the message m—the committer also computes an opening value, d, for the commitment that will be used to convince the verifier in the next stage.
3. Verification phase, The committer sends the verifier the message m and an opening value, d, with which the verifier can verify that the original committed message was m.

We formally model the three phases by fixing the types of three probabilistic programs (*key_gen*, *commit*, *verify*), seen in the locale given in Fig. 6.

The key generation algorithm outputs the keys available to the committer (*ck*) and the verifier (*vk*) respectively. If all the keys are public then we have $ck = vk$. We also fix two predicates abstractly which are needed in concrete instances later; *valid_msg* checks if a message is valid or not and $\mathcal{A}$*_cond* provides the conditions that we require from an adversary in the binding game. A paper proof can easily dismiss the adversary as failing if these conditions are not met,

```
locale abstract_com =
  fixes key_gen :: ('ck × 'vk) spmf
  and commit  :: 'ck ⇒ 'plain ⇒ ('com × 'open) spmf
  and verify  :: 'vk ⇒ 'plain ⇒ 'com ⇒ 'open ⇒ bool spmf
  and valid_msg  :: 'plain ⇒ bool
  and 𝒜_cond  :: 'com ⇒ 'plain ⇒ 'open ⇒ 'plain ⇒ 'open ⇒ bool
```

```
correct_game m = do {
  (ck, vk) ← key_gen;
  (c, d) ← commit(ck, m);
  b ← verify(vk, m, c, d);
  return b}
```

```
binding_game 𝒜 = TRY do {
  (ck, vk) ← key_gen;
  (c, m, d, m', d') ← 𝒜(ck);
  _ ← assert(𝒜_cond(c, m, d, m', d'));
  b ← verify(vk, m, c, d);
  b' ← verify(vk, m', c, d');
  return(b ∧ b')} ELSE return False
```

```
hiding_game (𝒜1, 𝒜2) = TRY do {
  (ck, vk) ← key_gen;
  ((m0, m1), σ) ← 𝒜1(vk);
  _ ← assert(valid_msg(m0) ∧ valid_msg(m1));
  b ← coin;
  (c, d) ← commit(ck, (if b then m1 else m2));
  b' ← 𝒜2(c, σ);
  return(b = b')} ELSE coin
```

Fig. 6. Abstract commitment scheme locale and definitions.

however formally we must catch this in the semantics. In fact these predicates serve another purpose too; they allow us to use the properties captured by the predicates in our reasoning at a later point in the proof. For example, for m to be a valid message we may require $m \in G$, this fact is then known to Isabelle for later use (e.g when applying Eq. 11).

6.1 Properties of Commitment Schemes

There are two main properties associated with commitment schemes: the hiding and binding properties. We note we consider a third property of correctness also, the need for this is explained at the end of the section.

Hiding. Intuitively, the hiding property is that no adversary can distinguish two committed messages. To define the hiding property we define the *hiding game* between an adversary, $\mathcal{A}$, and a benign challenger. The formal game can be seen in Fig. 6. The game asks the adversary to output two messages, one of which is committed to by the challenger and the corresponding commitment handed back to the adversary. The adversary is then asked to guess which message was committed to. The adversary wins the game if they correctly output which message was committed and handed to them.

Using the hiding game we can define the hiding advantage.

Definition 4. *The hiding advantage is the probability an adversary has of winning the hiding game.*

$$hiding\_advantage(\mathcal{A}) \equiv \mathcal{P}[hiding\_game(\mathcal{A}) = True]$$

Using this we can define perfect hiding, which holds for the Pedersen commitment scheme.

Definition 5. *For perfect hiding we require*

$$perfect\_hiding(\mathcal{A}) \equiv (hiding\_advantage(\mathcal{A}) = \frac{1}{2})$$

Binding. The binding property ensures that the committer cannot change her mind and change the message she has committed to. Again a security game is used (see Fig. 6). We challenge the adversary to bind two messages (m, m') and two opening values (d, d') to the same commitment c.

Similar to the hiding property we define the binding advantage:

Definition 6. *The binding advantage is the probability an adversary has of winning the binding game.*

$$\mathit{binding\_advantage}(\mathcal{A}) \equiv \mathcal{P}[\mathit{binding\_game}(\mathcal{A}) = \mathit{True}]$$

To show computational binding we must show the binding advantage is a negligible function with respect to the security parameter. This result can only be shown in the asymptotic setting as it requires an explicit security parameter. In the concrete setting we can show a reduction to a known hard problem (for the Pedersen scheme this is the discrete logarithm problem). We can then extend to the asymptotic setting. See Sect. 8.1 for more details on our proofs in the asymptotic setting.

Correctness. There is one subtlety to the binding definition meaning we must consider correctness also. If the verifier always outputs false, the binding property is met as the adversary will never win the game.

Correctness is the property that, assuming honest parties, a commitment will be verified as true by the verifier. Analogously to the hiding and binding properties we use the correctness game to define correctness.

Definition 7

$$\mathit{correct}(m) \equiv (\mathcal{P}[\mathit{correct\_game}(m) = \mathit{True}] = \mathit{1})$$

7 The Pedersen Commitment Scheme

In this section we discuss our formalisation of the Pedersen commitment scheme. We do not discuss the proofs in detail, but instead provide the formal results and a discussion of the interesting aspects learned from the proof.

The protocol, given in Fig. 7, is run using a cyclic group of prime order G with generator g.

Intuitively, the hiding property is observed because the message, m, is not sent explicitly, but is masked by the uniform sample, g^d (in $g^d.pk^m$). Consequently the verifier cannot distinguish between two committed messages. The property of binding is more subtle. If the adversary can bind two messages to the same committed value, then the adversary can also compute the discrete log of pk, which is in violation of the discrete log assumption which is considered hard. Correctness is immediate; the committed value, c, is $g^d.pk^m$, the verifier accepts the message if (m, d), sent by the committer is such that $g^d.pk^m = c$.

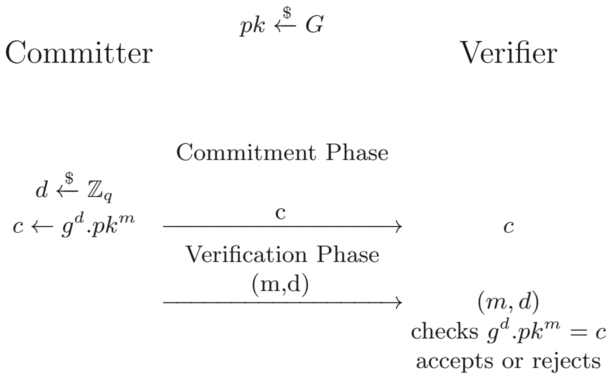

Fig. 7. The Pedersen commitment protocol.

7.1 Formal Proofs for the Pedersen Protocol

We fix a finite cyclic group, G, with generator, g, and order, $|G|$ and explicitly define the probabilistic programs that define the protocol.

Perfect Hiding. Lemma 3 shows that we have perfect hiding for the Pedersen commitment scheme.

Lemma 3. *For the Pedersen commitment scheme we have*

$$\mathcal{P}[(hiding\_game(\mathcal{A})) = True] = \frac{1}{2}$$

The security of the hiding property comes from applying the OTP lemma:

$$c \in carrier\ G \Rightarrow map((\lambda x.\ g^x \ \otimes\ c), (uniform(|G|))) = \\ map((\lambda x.\ g^x), uniform(|G|))). \tag{11}$$

The work needed to apply the one time pad Lemma is in showing that $c \in$ *carrier* G. To do this requires the use of some congruence lemmas as the property of membership of the carrier group arises from conditions on bound variables. Applying the one time pad Lemma shows the value given to the adversary is independent of m_b. Consequently the output from the adversary can be nothing more than a guess, in other words, the adversary may as well flip a coin to decide its output.

Computational Binding and Correctness. To prove the binding property we show a reduction to the discrete logarithm assumption. Hardness assumptions are a cornerstone of cryptography so we take a moment to consider how it may be formal used. One can follow a similar pattern for defining other hardness assumptions, for example see how the DDH assumption is defined in [16].

We first define the task of the adversary and then the advantage associated to the adversary. In the case of the discrete log assumption we simply provide the adversary with g^x where x is uniformly sampled and ask it to output x. We formalise such a situation as a game between the adversary and a challenger in Fig. 8.

```
$dis\_log\_game(\mathcal{A}) = do\ \{$
  $x \leftarrow sample\_uniform(|G|);$
  $let\ h = g^x;$
  $x' \leftarrow \mathcal{A}(h);$
  $return(x = x')\}$
```

Fig. 8. The discrete log game.

We then define the associated advantage of the adversary when playing this game—the probability of it winning the game.

Definition 8. $dis\_log\_advantage(\mathcal{A}) \equiv \mathcal{P}[(dis\_log\_game(\mathcal{A})) = True]$

To prove binding we construct an adversary, $dis\_log\_\mathcal{A}$, using the adversary $\mathcal{A}$ that plays the binding game and show it has the same advantage against the discrete log game as $\mathcal{A}$ has against the binding game. Our adversary here takes a similar form as that used in the proof of special soundness for the Schnorr protocol. Using this we can show Lemma 4 which easily shows Theorem 5.

Lemma 4

$$bind\_game(\mathcal{A}) = dis\_log\_game(dis\_log\_\mathcal{A}(\mathcal{A}))$$

Theorem 5

$$bind\_advantage(\mathcal{A}) = dis\_log\_advantage(\mathcal{A})$$

Finally we prove the correctness of the Pedersen scheme. This result comes easily after proving some group identities in Isabelle.

Theorem 6. *We have*

$$\mathcal{P}[(correct\_game(m)) = True] = 1.$$

8 Using Σ-Protocols to Construct Commitment Schemes

In [8], it was shown how commitment schemes can be constructed from Σ-protocols. One can use the components of a Σ-protocol, for a relation R, to form a commitment scheme as follows:

Key Generation. The keys are generated such that the verifier receives $(h, w) \in R$ that satisfy R and the committer receives only h.

Commit. The committer runs the simulator on their key h and the message, m, they wish to commit. That is they run

$$(a, e, z) \leftarrow S(h, m)$$

and sends a to the verifier and keeps e and z as the opening values.

Verify. In the verification stage the prover sends e and z to the verifier who uses the check algorithm of the Σ-protocol to confirm that (a, e, z) is an accepting conversation, with respect to the private key w.

We recall that in the *Commit* stage we have $e = m$. The resulting commitment scheme can be shown to be perfectly hiding and computationally binding. Intuitively perfect hiding comes from the fact that the simulation is perfect (the simulated and real views are equal) and that the initial message is not dependent on the challenge. Binding holds as if a prover could output two sets of opening values, (e, z) and (e', z'), for one commitment a such that $e \neq e'$ then (a, e, z) and (a, e', z) would both be accepting conversations and by the special soundness property we could compute w, but this contradicts the hardness assumption on R. We formally prove these results in Isabelle for the commitment scheme constructed from the Schnorr Σ-protocol.

To formalise this in Isabelle we again explicitly define the constants required for commitment schemes. We define these using the constants defined for the Schnorr Σ-protocol. This requires us to import both locales (for commitment schemes and Σ-protocols) and prove all assumptions relating to them. We are then able to prove perfect hiding, computational binding and correctness of the resulting commitment scheme. In the concrete setting we show a reduction of the binding property to the discrete logarithm assumption.

Theorem 7. *For the commitment scheme constructed from the Schnorr protocol we have*

$$\mathcal{P}[correct\_game(m) = True] = 1$$

$$\mathcal{P}[hiding\_game(\mathcal{A}) = True] = \frac{1}{2}$$

$$\mathcal{P}[bind\_game(\mathcal{A}) = True] = \mathcal{P}[dis\_log(dis\_log\_\mathcal{A}(\mathcal{A})) = True]$$

This result has taken an instantiated Σ-protocol, used its components to instantiate a commitment scheme and proven this secure with respect the definitions we formalised for commitment schemes.

8.1 Asymptotic Case

So far in our formalisation the security parameter has been assumed to be implicit in all algorithms (probabilistic programs). In this section we show how we formalise proofs in the asymptotic setting using as an example the commitment scheme we have just constructed using the Schnorr Σ-protocol. In our formalisation we provide proofs in the asymptotic setting for all instantiations.

The asymptotic setting is particularly interesting in the case of commitment schemes as we can consider computational binding; a full proof will show the adversary has only negligible chance of winning the binding game.

To realise such a proof we parametrise over the family of cyclic groups, specifically we change the type from *'grp cyclic_group* to *nat* $\Rightarrow$ *'grp cyclic_group*. Thus the cyclic group is parametrised by the security parameter—a natural. After importing the concrete setting parametrically for all n, all algorithms now depend explicitly on the security parameter. Moreover, due to Isabelle's module structure we are able to use results proven in the concrete setting in our newly constructed asymptotic setting. It is worth noting that results from the concrete setting can only be used once it has been proven that the import is valid, something the user is required to do when importing a module.

The properties, in the asymptotic case, for correctness and hiding can be seen in Theorem 8. Superficially, the only difference is the security parameter is an input to every statement and function. At a deeper level the proof uses the equivalent theorems from the concrete setting and the module machinery to dismiss the proof.

Theorem 8. *In the asymptotic case, for security parameter, n, we have:*

- $\mathcal{P}[\mathit{correct\_game}(n, \mathit{msg}) = \mathit{True}] = 1$
- $\mathcal{P}[\mathit{hiding\_game}(n, (\mathcal{A}(n))) = \mathit{True}] = \frac{1}{2}$.

The more interesting case is the proof of computational binding as we are required to show the binding advantage is negligible. In the concrete setting (Theorem 7) we showed we could construct an adversary that had the same advantage against the discrete log problem as the binding game. In the asymptotic setting we are able to assume that the discrete logarithm assumption holds; that an adversary only has a negligible chance of winning the discrete log game. Using this we can prove that the binding advantage too is negligible. This is shown in Theorem 9.

Theorem 9

$$\begin{aligned}&\mathit{negligible}\ (\lambda n.\ \mathit{bind\_advantage}\ n\ (\mathcal{A}\ n)) \iff \\ &\qquad\qquad \mathit{negligible}\ (\lambda n.\ \mathit{dis\_log\_advantage}\ n\ (\mathit{dis\_log\_}\mathcal{A}\ n\ (\mathcal{A}\ n)))\end{aligned}$$

Our formalisation provides proofs in the asymptotic case for all relevant properties presented in this paper in a similar manner as described above. We refer the reader to our formalisation for more details.

9 Conclusions

In this work we have demonstrated that commitment schemes and Σ-protocols can be formally proved secure in the computational model using a general abstract framework. Our work uses Isabelle/HOL and its modularity mechanisms, but in principle could be replicated in other interactive theorem provers.

The abstract frameworks can be used by others to formalise new commitment schemes and Σ-protocols. The advantages of reasoning back to the same general framework is that one can be sure the correct properties and definitions are being considered. This consistency is not always apparent in informal cryptographic proofs. We suggest that cryptographic advances should be monitored within a formal framework where one is required to use the exact definitions set out formally (the proof could be done on pen and paper) or provide a formal proof that the chosen definitions are equivalent. This will help alleviate the abundance of small differences in definitional approaches which undermine the field.

At the present state-of-the-art, prototyping this approach in an interactive theorem prover seems essential as it allows one to explore the reasoning and definition principles which are most effective in the domain. Eventually we may hope that bespoke foundational reasoning tools could be built which may be more usable by applied cryptographers (as is the aim of EasyCrypt, although it is not foundational).

One major advantage of our framework being implemented in Isabelle is that we can benefit from the vast infrastructure that comes with a well developed theorem prover, in contrast with custom made tools. We benefit from the interactive nature of Isabelle meaning users have flexibility but also the high level of automation and many proof engines available.

While CryptHOL and thus our framework still require a high level of specific interactive theorem proving knowledge to use, new features are being developed that make it more usable by the working cryptographer. For example recent work in Isabelle [19], monad normalisation, has made handling the commuting of samplings, a previously technical and subtle exercise, more simple. As more features like this automate the intricate details needed in proofs the barrier to entry to using CryptHOL will be significantly reduced.

Future Work. The frameworks we provide here can be used and instantiated to give formal proofs of new commitment schemes and Σ-protocols. Both of these primitives are used to provide security in the malicious model, consequently we see this work as a building block to further formal proofs here.

10 Related Work

Little work has been done on formalising the computational model, compared to the symbolic model. It is challenging as it requires mathematical reasoning about probabilities and failure events, besides logic. The CertiCrypt [3] tool built in Coq helped to capture the reasoning principles that were implemented directly in the dedicated interactive EasyCrypt tool [4]. Again in Coq, the Foundational Cryptographic Framework [18] 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.

CryptHOL [6], formalised in Isabelle, has been used in the game-based setting [16], as well as the simulation-based paradigm [7]. Isabelle is a foundational framework that relies on a set of accepted consistent axioms [10,13] and thus provides a high guarantee of correctness in proofs.

The Pedersen commitment scheme has been proved secure in EasyCrypt in [17]. One noticeable difference between the proof effort required is in the construction of the adversary used to prove computational binding. We had to work hard in Isabelle to give the output of the adversary in the binding game as a division of two elements in the finite field. We are required to prove extra properties of the Bezout function whereas the division can be easily expressed in EasyCrypt.

10.1 Comparison with EasyCrypt

EasyCrypt is considered the state of the art in terms of proof assistants for cryptography; it was designed as a dedicated tool for the working cryptographer. It has a larger user base than CryptHOL, partially due to it having been developed a number of years before and its greater support and documentation. The barrier to entry to using EasyCrypt is lower in comparison to Isabelle. We argue however that there is room for more than one proof assistant when considering cryptographic proof; in fact we suggest that it is essential to the development of formal proof in this area. Growth in an area of research is rarely achieved by considering only one approach; different proof assistants allow for different proof styles and thus different insights into the cryptographic proofs at a fundamental level.

One such difference in approach is the ability to follow paper proofs explicitly. Isabelle's deep and extensive foundations in mathematical logic meaning there is a large amount of machinery available to the user when completing proofs. This allows one to more closely follow the proof method given in the paper proof. In EasyCrypt sometimes this is not possible. For example in [11] the authors had to prove on paper that the security definitions they formalised were equivalent to the traditional definitions in the literature. At a technical level this is because the proof technique in EasyCrypt is often to reduce proofs to showing properties about the equivalence of programs. This is not necessarily a weakness of EasyCrypt, as it allows for new insights into proof techniques but it highlights a difference between the two systems.

Finally, when using CryptHOL in Isabelle all proofs are checked with respect the same logical core. That is, the whole CryptHOL framework resides within Isabelle. However, some fundamental properties in EasyCrypt are outsourced to be proven in Coq. Thus one could consider the approach of CryptHOL and Isabelle to be more foundational.

Acknowledgements. We are grateful to Andreas Lochbihler for providing and continuing to develop CryptHOL and for his kind help given with using it.

References

1. https://github.com/alan-turing-institute/Crypto-Commit-Sigma-Isabelle
2. Bacelar Almeida, J., Barbosa, M., Bangerter, E., Barthe, G., Krenn, S., Zanella Béguelin, S.: Full proof cryptography: verifiable compilation of efficient zero-knowledge protocols. In: ACM Conference on Computer and Communications Security, pp. 488–500. ACM (2012)
3. Barthe, G., Grégoire, B., Zanella Béguelin, S.: Formal certification of code-based cryptographic proofs. In: POPL, pp. 90–101. ACM (2009)
4. Barthe, G., Grégoire, B., Heraud, S., Zanella Béguelin, S.: Computer-aided security proofs for the working cryptographer. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 71–90. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_5
5. Barthe, G., Hedin, D., Zanella Béguelin, S., Grégoire, B., Heraud, S.: A machine-checked formalization of sigma-protocols. In: CSF, pp. 246–260. IEEE Computer Society (2010)
6. Basin, D.A., Lochbihler, A., Sefidgar, S.R.: CryptHOL: game-based proofs in higher-order logic. IACR Cryptology ePrint Archive, p. 753 (2017)
7. Butler, D., Aspinall, D., Gascón, A.: How to simulate it in Isabelle: towards formal proof for secure multi-party computation. In: Ayala-Rincón, M., Muñoz, C.A. (eds.) ITP 2017. LNCS, vol. 10499, pp. 114–130. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66107-0_8
8. Damgard, I.: On Σ-protocols. Lecture Notes, University of Aarhus, Department for Computer Science (2002)
9. Damgård, I.: Commitment schemes and zero-knowledge protocols. In: Damgård, I.B. (ed.) EEF School 1998. LNCS, vol. 1561, pp. 63–86. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48969-X_3
10. Gordon, M.J.C., Melham, T.F.: Introduction to HOL a Theorem Proving Environment for Higher Order Logic. Cambridge University Press, New York (1993)
11. Haagh, H., Karbyshev, A., Oechsner, S., Spitters, B., Strub, P.-Y.: Computer-aided proofs for multiparty computation with active security. In: CSF, pp. 119–131. IEEE Computer Society (2018)
12. Halevi, S.: A plausible approach to computer-aided cryptographic proofs. IACR Cryptology ePrint Archive 2005:181 (2005)
13. Kunčar, O., Popescu, A.: A consistent foundation for Isabelle/HOL. In: Urban, C., Zhang, X. (eds.) ITP 2015. LNCS, vol. 9236, pp. 234–252. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22102-1_16
14. Lochbihler, A.: CryptHOL. Archive of Formal Proofs (2017)
15. Lochbihler, A.: Probabilistic functions and cryptographic oracles in higher order logic. In: Thiemann, P. (ed.) ESOP 2016. LNCS, vol. 9632, pp. 503–531. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49498-1_20
16. Lochbihler, A., Sefidgar, S.R., Bhatt, B.: Game-based cryptography in HOL. Archive of Formal Proofs (2017)
17. Metere, R., Dong, C.: Automated cryptographic analysis of the Pedersen commitment scheme. In: Rak, J., Bay, J., Kotenko, I., Popyack, L., Skormin, V., Szczypiorski, K. (eds.) MMM-ACNS 2017. LNCS, vol. 10446, pp. 275–287. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65127-9_22
18. Petcher, A., Morrisett, G.: The foundational cryptography framework. In: Focardi, R., Myers, A. (eds.) POST 2015. LNCS, vol. 9036, pp. 53–72. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46666-7_4

19. Schneider, J., Eberl, M., Lochbihler, A.: Monad normalisation. Archive of Formal Proofs (2017)
20. Smart, N.P.: Cryptography Made Simple. Information Security and Cryptography. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-21936-3. https://www.cs.umd.edu/~waa/414-F11/IntroToCrypto