Formalising Σ-Protocols and Commitment Schemes using CryptHOL

D. Butler · A. Lochbihler · D. Aspinall · A. Gascón

Abstract Machine-checked proofs of security are important to increase the rigour of provable security. In this work we present a formalised theory of two fundamental two party cryptographic primitives: Σ-protocols and Commitment Schemes. Σ-protocols allow a prover to convince a verifier that they possess some knowledge without leaking information about the knowledge. Commitment schemes allow a committer to commit to a message and keep it secret until revealing it at a later time.

We use CryptHOL [31] to formalise both primitives and prove secure multiple examples namely; the Schnorr, Chaum-Pedersen and Okamoto Σ-protocols as well as a construction that allows for compound (AND and OR) Σ-protocols and the Pedersen and Rivest commitment schemes.

A highlight of the work is a formalisation of the construction of commitment schemes from Σ-protocols [22]. We formalise this proof at an abstract level using the modularity available in Isabelle/HOL and CryptHOL. This way, the proofs of the instantiations come for free.

1 Introduction

Provable security provides a firm mathematical foundation for reasoning about cryptography. A variety of definition styles have been proposed to reason about

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

D. Butler
Alan Turing Institute, 96 Euston Road, London, UK
E-mail: dbutler@turing.ac.uk

A. Lochbihler
Digital Asset (Switzerland) GmbH, Thurgauerstrasse 40, 8050 Zurich, Switzerland
E-mail: mail@andreas-lochbihler.de

D. Aspinall
University of Edinburgh, Edinburgh, UK
E-mail: david.aspinall@ed.ac.uk

A. Gascón
Google (London), 6 Pancras Square, London, UK
E-mail: adriagascon@gmail.com
Note 1 This work was done while A.G was at Warwick University and the Alan Turing Institute.

security in different settings. For example, simulation-based definitions [15,26] capture the security notions in *Multi-Party Computations* (MPC), and game-based definitions [7,38] formalise the security of primitives like *encryption* and *commitments.*

Security proofs are now a cornerstone of modern cryptography. Provable security has greatly increased the level of rigour of the security statements, however proofs of these statements often present informal or incomplete arguments. In fact, many proofs are still considered to be *unverifiable* [7,28]. Formal methods offer one way to establish far higher levels of rigour in proofs and tools have been developed to formally reason about cryptography and obtain machine-checked proof of security statements. Formalisation of cryptography is a maturing area of research; the EasyCrypt framework [2] has captured proofs of low-lying cryptographic primitives [33] as well as MPC [27] and Universal Composibility [16]. Moreover CryptHOL [30] has also considered fundamental primitives [13,30] and MPC protocols [11,12] as well as Constructive Cryptography [32]. Other tools for reasoning about cryptographic proofs in the context of our work include FCF [35], which provides a shallow embedding in Coq for reasoning about cryptography and CertiCrypt [1], a deep embedding in Coq in which the first (and only, before this work) formalisation of Σ-protocols was made [5].

In this work we consider two fundamental cryptographic primitives, namely Σ-protocols and commitment schemes, and their connection. Commitment schemes allow a party to commit to a message and keep it hidden until it is chosen to be revealed at a later time. In particular commitment schemes are used to hold parties accountable to the messages they send; ensuring they do not *cheat* when participating in protocols. To this end, commitments are often used to extend protocols secure in the semi-honest model (where parties are assumed to follow the protocol) to be secure in the malicious setting (where corrupted parties may arbitrarily deviate from the protocol).

Σ-protocols allow for a party, the prover, to convince a verifier they possess some knowledge. More formally, we consider a relation R and say w is a witness to the relation with respect public input x if $(x, w) \in R$. A Σ-protocol allows the prover to convince the verifier that the prover knows w for some given x without revealing anything else about w itself. Like commitment schemes, Σ-protocols aid the enforcement of honest behaviour from potentially malicious parties. For example the witness (and proof of knowledge of the witness) can provide a guarantee that the party is authorised to perform certain actions, or access certain sensitive information.

The two primitives are strongly linked; Damgård [22] showed how Σ-protocols can be used to construct commitment schemes. So every Σ-protocol yields a corresponding commitment scheme.

Our formalisation is done using the CryptHOL framework inside Isabelle/HOL. We have chosen CryptHOL for three reasons: First, it provides the expressiveness and rigour of higher-order logic. Second, we believe the resulting formalisations are easy to read, even for the non formal methods expert; this is something we feel is important. Third, it supports different styles of security definitions. The security of commitment schemes is expressed using game-based definitions whereas Σ-protocols' security definitions contain a flavour of the simulation-based proof method. Therefore our work draws on the originally designed application of CryptHOL (game-based proofs) [30] as well as more recently considered applications (simulation-based proofs) [12].

Contributions By leveraging the expressiveness and modularity of CryptHOL and Isabelle we develop a framework for formally reasoning about the security proofs of commitment schemes and Σ-protocols. To the best of our knowledge this is the first formalisation that links the two primitives.

1. We formalise a framework for reasoning about the security of commitment schemes and Σ-protocols in a general manner. This provides an abstract basis for others to use as well as lends weight to the notion that CryptHOL is an appropriate framework for cryptography. Out of the various Σ-protocol definitions in the literature, we identify Cramer's definition from his PhD thesis [20] as the right one. In particular, we highlight that the standard textbook definition [29] and Damgård's [22] are too weak.
2. We demonstrate how our general frameworks can be instantiated by proving security of well-known examples of both primitives. In detail, we formalized the Σ-protocols by Schnorr, Chaum-Pedersen, and Okamoto; and the commitment schemes by Rivest and Pedersen.
3. We prove the construction of commitment schemes from Σ-protocols [22] secure at an abstract level. That is, the construction works for any Σ-protocol. Consequently the proof effort for any instantiations of the construction is only in proving that the underlying Σ-protocol is secure. The commitment scheme result then comes in a matter of lines of proof. At an estimate this halves the proof effort as, in our experience, proofs of commitment schemes' security are similar in length (and effort) to proofs of Σ-protocols. In particular, for every new Σ-protocol proven secure in our framework we get a proof of a new commitment scheme being secure for free. For example, security for the Pedersen commitment scheme needs about 20 proof lines compared to a few hundred in previous work [13].
4. We formalise the AND and OR compound statement construction of two Σ-protocols. Here we generalise the proof to arbitrary boolean algebras. The construction from the literature [21] given over bitstrings is one instance of our result.

This paper extends and improves the conference paper [13] as follows:

- We additionally formalize and prove secure the Rivest commitment scheme and the Okamoto Σ-protocol. The Rivest commitment scheme uses a trusted initialiser who distributes correlated randomness to both parties. Formalising this result shows that our framework can cope with different structures of commitment scheme.
- We formalize the generic construction of commitment schemes from Σ-protocols. In [13], only the instantiated results were formalised.

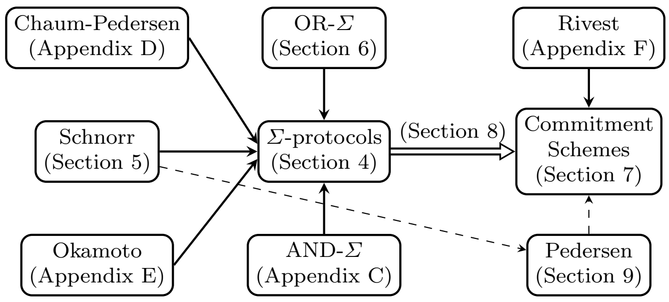

Fig. 1: The diagram outlines our formalisation in this paper.

– The formalisation of compound statements of two Σ-protocols is new to this work.

Outline Figure 1 outlines the work we present in this paper. Solid arrows represent proofs of concrete commitment schemes or Σ-protocols; the arrows end at the instantiated framework. The double arrow represents our formalisation of the general construction of commitment schemes from Σ-protocols, and the corresponding commitment schemes from our instantiated Σ-protocols whose security statements come for free due to the general proof. We highlight one of these in particular with the dotted arrow as the instantiation of the Schnorr Σ-protocol under the general construction yields the Pedersen commitment scheme, a result we formalised from scratch in [13] but comes in a matter of lines of proof here.[1]

In Section 2 we introduce the relevant background on Σ-protocols, commitment schemes, and CryptHOL. Section 3 outlines the general method of formalising cryptographic primitives in CryptHOL. In Sections 4 and 7 we introduce our formalisation of Σ-protocols and commitment schemes respectively. We show how they can be instantiated for the Schnorr Σ-protocol in Section 5, compound statements of Σ-protocol relations in Section 6, and for the general proof of commitment schemes from Σ-protocols in Section 8. We show in Section 9 how the security of the Pedersen commitment scheme follows from the general proof. We discuss related work in Section 11 and detail how, during our formalisation, we came across discrepancies in the definitions of Σ-protocols and how we resolved this. Finally in we conclude in Section 12.

The security definitions presented in Section 2.1 are the traditional paper-based definitions of commitment schemes and Σ-protocols; all definitions and statements given in the rest of the paper have been checked by the proof assistant Isabelle/HOL.

[1] Our formal proofs can be found at [14].

2 Background

2.1 Σ-protocols and Commitment Schemes

Commitment schemes and Σ-protocols are two party protocols considered to be fundamental building blocks in modern cryptography. Commitment schemes allow a party to commit to a message and reveal it at a later time. This is a powerful construction that is widely used, for example in MPC where they are used as a tool to convert semi-honest protocols to protocols secure in the stronger malicious model. Σ-protocols allow a prover to convince a verifier of some knowledge they posses and are a direct building block for Zero-Knowledge proofs. The major limitation of Σ-protocols is that they do not account for a cheating verifier, it is assumed that the verifier follows the protocol exactly — this is analogous to the semi-honest model considered in simulation-based proofs.

2.1.1 Σ-protocols

Cramer [20] introduced the abstract notion of a Σ-protocol, coined the term Σ-protocol, and gave the definitions of the properties we consider here. He also developed a rich theory of Σ-protocols that goes beyond what we formalise in this work. Schnorr introduced the first efficient Σ-protocol [37] — the protocol we formalise in Section 5. The presentation of Σ-protocols follows Damgård [22], Hazay and Lindell [29] and Cramer [20].[2]

A Σ-protocol is considered with respect to a relation R. If $(h, w) \in R$ then h can be considered an instance of a computational problem where w is the witness or solution to the problem. For example consider the discrete log relation which is considered over a group G with generator g. We say w is a witness to $h \in G$ if the following relation holds.

$$(h, w) \in R_{DL} \iff h = g^w \tag{1}$$

The discrete log relation is widely used in cryptography as for certain groups (e.g $\mathbb{Z}_p^*$ and elliptic curves over finite fields) it is considered a hard relation, that is it is computationally infeasible to obtain the witness w from $h = g^w$.

Any relation, R, gives rise to a language $L_R = \{h.\ \exists w.\ (h, w) \in R\}$ that consists of statements in R.

A Σ-protocol is a three move protocol run between a Prover (P) and a Verifier (V) where h is common public input to both P and V and w is a private input to P such that $(h, w) \in R$.

Definition (informal) 1 *A Σ-protocol has the following three part form:*

[2] Damgard's [22] and Hazay's and Lindell's definitions [29] are too weak. Our definition of a Σ protocol in Definition 2 therefore includes Cramer's additional requirements. A detailed discussion can be found in Section 11.1.

$$\begin{array}{ccc}
\textit{Prover} & & \textit{Verifier} \\
(a, r) \leftarrow \textit{initial} & \xrightarrow{\quad a \quad} & a \\
e & \xleftarrow{\quad e \quad} & e \leftarrow \textit{challlenge} \\
z \leftarrow \textit{response} & \xrightarrow{\quad z \quad} & z \\
 & & \textit{accepts/rejects}
\end{array}$$

That is: first the Prover sends an initial message a which is created using randomness r (sampled by the Prover), second the Verifier sends a challenge e and finally the Prover sends a response, from which the Verifier decides if it will accept or reject the proof.

A *conversation* for an execution of a Σ-protocol is the transcript of the protocol — (a, e, z). The conversation is said to be accepting if the tuple corresponds to the outputs of the three moves in the protocol and the verifier accepts the response z.

There are three properties that are required for a protocol of the above form to be a Σ-protocol.

Definition (informal) 2 *Assume a protocol, π, of the above form run between P and V. Then π is a Σ-protocol for a relation R if the following properties hold:*

- *Completeness: if P and V follow the protocol on public input h and private input w such that $(h, w) \in R$, then V always accepts.*
- *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 $(h, w) \in R$.*
- *Honest verifier Zero-Knowledge (HVZK): The following conditions must hold.*
  1. *There exists a polynomial-time simulator S that on input h (public input) and e (a challenge) outputs an accepting conversation (a, e, z) with the same probability distribution as the real conversations between P and V on input (h, w). That is for all h and w such that $(h, w) \in R$ and every e we have*
  $$\{S(h, e)\} = \{\langle P(h, w), V(h, e)\rangle\}$$
  where $\{S(h, e)\}$ is the output distribution of the simulator and $\{\langle P(h, w), V(h, e)\rangle\}$ denotes the distribution of the output transcript of an execution of the protocol between P and V.
  2. *For $h \notin L_R$ the simulator $S(h, e)$ must nevertheless output an accepting conversation (a, e, z).*

Completeness provides a notion of correctness for the protocol, that is if the protocol is executed honestly then the Verifier will accept. The intuition for the special soundness property is that if a Prover can respond correctly to two different challenges then it can also compute the witness, meaning it is infeasible for a Prover to cheat a Verifier — that is convince the verifier when a witness is not known to the prover. The HVZK property ensures that no information about the witness is leaked during the execution of the protocol. The first condition resembles definitions from Multi-Party Computation (MPC) where the real view (the real conversation generated by the Prover and Verifier) can be simulated without the private input (the witness). Condition 2 ensures that the OR construction of Σ-protocols satisfies completeness (Section 6.1).

2.1.2 Commitment Schemes

Commitment schemes were first introduced by Blum [8] and Even [25]. The problem Blum proposed was that of coin flipping by telephone; how do Alice and Bob flip a coin via telephone. Blum proposed commitments to solve such a problem: Alice *calls* the coin flip and commits to her call, Bob then flips the coin and reveals the result upon which Alice reveals the value she committed to so Bob can verify her call matches her commitment — if Alice's call matches the coin flip she wins.

Definition (informal) 3 *A commitment scheme has the following three part form:*

1. *Key generation:* $(ck, vk) \leftarrow key$*. The algorithm key outputs a pair of keys that is sent to the committer and verifier respectively.*
2. *Commitment phase:* $(c, d) \leftarrow com(ck, m)$*. The algorithm com takes as input the message to be committed and outputs the commitment c and an opening value d, which is sent to* V *in the verification phase.* C *sends c to* V*.*
3. *Verification phase:* $b \leftarrow ver(vk, c, m, d)$*. The algorithm ver takes the verification key, commitment, original message and opening value as input and outputs a boolean depending on whether the verification is successful.*

The three properties we want from a commitment scheme are correctness, hiding and binding.

Definition (informal) 4 (Correctness) *A commitment scheme is said to be correct if the protocol is run honestly between* C *and* V*, then* V *will always accept in the verification phase for all messages that can be committed.*

To define the hiding and binding properties cryptographers consider security games that are played between an adversary and a benign challenger. Games are used to *tame complexity* [38] of security proofs. The security games we consider can be considered as pseudo protocols played between the committer and the verifier, where one of the parties is controlled by an adversary and the other is the challenger. Consider the hiding game depicted in Figure 2. Here the committer is the challenger and the verifier the adversary; the keys are distributed and the adversary asked to output two messages of its choosing and send them to the committer upon which the committer picks one at random and constructs its commitment. The adversary is then required

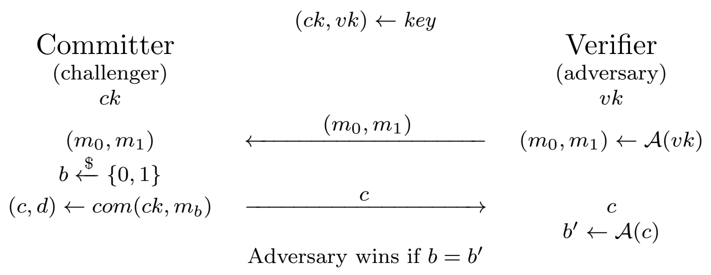
$(ck, vk) \leftarrow key$
Committer
(challenger)
ck
Verifier
(adversary)
vk
(m_0, m_1)
(m_0, m_1)
$(m_0, m_1) \leftarrow \mathcal{A}(vk)$
$b \overset{\$}{\leftarrow} \{0, 1\}$
$(c, d) \leftarrow com(ck, m_b)$
c
c
$b' \leftarrow \mathcal{A}(c)$
Adversary wins if $b = b'$

Fig. 2: The hiding game played between the committer (the challenger) and the adversary (the verifier).

to output its guess as to which message was committed and wins the game if it guesses correctly. More generally the definition of security with respect to a security game is tied to an event E (in the hiding game this is $b = b'$), security requires that the probability that E occurs *close* to some target probability (this is $\frac{1}{2}$ for the hiding property) — the difference between the probability of the event E occurring and target probability is called the advantage of the adversary. Intuitively security is achieved if this advantage is small.

The game-based approach allows the cryptographer to be more formal in their reasoning about security properties. In particular they afford the opportunity to provide more rigorous proofs of security. A proof is generally structured as follows: let $G_0, \dots, G_n$ be a sequence of games where G_0 is the original security game and G_n is a game where the target probability is met. In the proof one shows that $|Pr[G_i] - Pr[G_{i+1}]|$ is small and thus the value of $|Pr[G_0] - Pr[G_n]|$ is also small.

We note that all the definitions here are actually parameterised by a security parameter and it must be shown that the advantage approaches zero faster than any inverse polynomial grows — that is the advantage is a negligible function. In our presentation here we omit the security parameter and refer only to the advantages of adversaries. Intuitively the security parameter gives a measure of the level of security of the protocol, a higher security parameter means a higher level of security. Practically this is realised by, for example, the size of group or field the protocol is considered over.

To define the hiding property we consider the algorithm which plays out the hiding game from Figure 2. Informally the algorithm, *hid-game*, is as follows:

```
1. $(ck, vk) \leftarrow key$
2. $(m_0, m_1) \leftarrow \mathcal{A}(vk)$
3. $b \overset{\$}{\leftarrow} \{0, 1\}$
4. $(c, d) \leftarrow com(ck, m_b)$
5. $b' \leftarrow \mathcal{A}(c)$
6. $return\ b = b'$
```

The notation $\overset{\$}{\leftarrow}$ denotes uniform sampling while we use $\leftarrow$ to denote assignment.

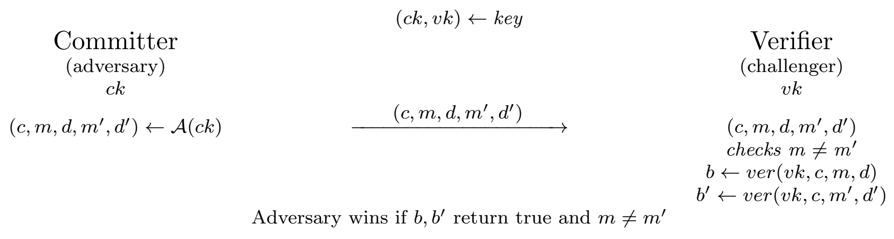
$(ck, vk) \leftarrow key$
Committer
(adversary)
ck
$(c, m, d, m', d') \leftarrow \mathcal{A}(ck)$
(c, m, d, m', d')
Verifier
(challenger)
vk
(c, m, d, m', d')
$checks\ m \neq m'$
$b \leftarrow ver(vk, c, m, d)$
$b' \leftarrow ver(vk, c, m', d')$
Adversary wins if b, b' return true and $m \neq m'$

Fig. 3: The binding game played between the challenger (the verifier) and the adversary (the committer).

Definition (informal) 5 (Hiding) *The hiding advantage is defined for all polynomial-time adversaries, $\mathcal{A}$, as*

$$hid\text{-}adv(\mathcal{A}) = |Pr[hid\text{-}game(\mathcal{A}) = 1] - \frac{1}{2}|$$

The scheme is said to be perfectly hiding if for all adversaries, $\mathcal{A}$, we have

$$hid\text{-}adv(\mathcal{A}) = 0.$$

The scheme is said to be computationally hiding if for all computationally bounded adversaries, $\mathcal{A}$, the advantage value $hid\text{-}adv(\mathcal{A}) - \frac{1}{2}$ is negligible. [3]

Analogously to the hiding property we define the binding property respect to the binding game which is depicted in Figure 3. The informal algorithm for playing the binding game is as follows:

1. $(ck, vk) \leftarrow key$
2. $(c, m, d, m', d') \leftarrow \mathcal{A}(ck)$
3. $checks\ m \neq m'$
4. $b \leftarrow ver(vk, c, m, d)$
5. $b' \leftarrow ver(vk, c, m', d')$
6. $return(b' \wedge b)$

Intuitively the challenger asks the adversary to output two messages (m, m') and corresponding opening values (d, d') for the same commitment c. If the adversary can achieve this such that both messages (and corresponding opening values) verify then the adversary (the committer) is not *bound* to the original message they commit to.

Definition (informal) 6 (Binding) *The binding advantage is defined for all polynomial-time adversaries, $\mathcal{A}$, as*

$$bind\text{-}adv(\mathcal{A}) = Pr[bind\text{-}game(\mathcal{A}) = 1]$$

[3] Computational bounds and negligibility are typically used in asymptotic security statements. There, all definitions are parametrised by a security parameter η and an adversary's run-time must be bounded by a (polynomial) function of η. Then, the advantage is negligible if it approaches 0 faster than any inverse polynomial as the security parameter grows.

The scheme is said to be perfectly binding if for all adversaries, $\mathcal{A}$, we have

$$\textit{bind-adv}(\mathcal{A}) = 0.$$

The scheme is said to be computationally binding if for all computationally bounded adversaries, $\mathcal{A}$, the advantage bind-adv$(\mathcal{A})$ is negligible.

We revert back to our coin flipping example to give some intuition regarding these properties. In the example Alice is the committer and Bob the verifier. Firstly we want the scheme to be correct, that is if both parties run the commitment protocol in the prescribed way then the Verifier will always be convinced in the verification phase. Secondly, we do not want Bob to be able to learn anything about Alice's call (what she commits to) from the commitment itself — that is we want the commitment to be hiding. Finally we do not want Alice to be able to decommit to a different call of the coin flip from the one she committed to, that is we want her commitment to be binding.

2.1.3 Commitments from Σ-protocols

Damgard [22] 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. Let the underlying Σ-protocol be π. Then the idea of the construction is that the initial message, a, of the Σ-protocol acts as the commitment where the challenge, e, is the committed message and, z, the response, is the opening value. To verify the commitment the verifier checks that (a, e, z) is an accepting conversation with respect to π.

In particular, this construction allows for a new secure commitment scheme for every Σ-protocol that is proved secure.

2.2 CryptHOL and Isabelle Background

In this section we introduce Isabelle/HOL and CryptHOL highlighting the parts important to our work. For more detail on CryptHOL see [6, 30, 31].

2.2.1 Isabelle notation

Isabelle/HOL is an interactive theorem prover that implements Higher Order Logic (HOL). HOL is built on simple set-theory, where types are interpreted as sets of elements and terms are elements of the set corresponding to their type. In this section we highlight some of the basic notions and notations we use in this paper, however for a more comprehensive overview we point the reader to [34].

The notations we use in this paper resemble closely the syntax of Isabelle/HOL (Isabelle)[4]. For function application we write $f(x, y)$ in an un-

[4] Figures 11 and 12 display the actual Isabelle code of the instantiation of the Pedersen commitment scheme and the corresponding asymptotic security statements. They therefore do not adhere to the slightly simplified notation used in the rest of the paper.

curried form for ease of reading instead of $f\ x\ y$ as in the sources. 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 variable $'a$ denotes an abstract type. The implication arrow $\longrightarrow$ is used to separate assumptions from conclusions inside a HOL statement. In HOL a function may be nameless, that is, $\lambda x.\ s(x)$, is the function that maps every value w to the results of s where x is replaced by w. In the situation where s does not depend on x, the underscore $\_$, replaces x in our notation. Pairs have the type $'a \times 'b$, the projections of the first and second elements are written *fst* and *snd* respectively.

One technical aspect of Isabelle we use heavily is the module system, called locales. At a technical level locales allow the user to prove theorems abstractly, relative to given assumptions. These theorems can be reused in situations where the assumptions themselves are theorems. For example we use locales to parametrise over cyclic groups as well as fix parameters and assumptions. The locale system also allows us to modularise our proofs in a natural way; we expand on this in Section 3.1.

2.2.2 CryptHOL

CryptHOL [6] is a framework for reasoning about *reduction-based* security arguments that is embedded inside the Isabelle/HOL theorem prover. At a high level it allows the user to formally reason about game-based cryptographic proofs by writing probabilistic programs and reason about relationships between them.

CryptHOL, like much of modern cryptography, is based on probability theory. Probabilistic programs in CryptHOL are shallowly embedded as subprobability mass functions of type *spmf* using Isabelle's library for discrete distributions. These can be thought of as probability mass functions with the exception that they do not have to sum to one — we can lose some probability mass. This allows us to model failure events and assertions. When a sub probability mass function does sum to one, we say it is lossless.

HOL functions cannot in themselves provide effects like probabilistic choice therefore all such effects are modeled using monads. A monad consists of a (polymorphic) type constructor, in this case *spmf* and two (polymorphic) operations, $\mathit{return}\ ::\ \alpha \Rightarrow \alpha\ \mathit{spmf}$ and $\mathit{bind}\ ::\ \alpha\ \mathit{spmf} \Rightarrow (\alpha \Rightarrow \beta\ \mathit{spmf}) \Rightarrow \beta\ \mathit{spmf}$.

We now introduce the parts of CryptHOL that are relevant for this paper.

Writing probabilistic programs Probabilistic programs can be encoded as sequences of functions that compute over values drawn from spmfs. CryptHOL provides some easy-to-read do notation, like in Haskell, to write probabilistic programs, where $\mathbf{do}\{x \leftarrow p;\ f(x)\}$ is the probabilistic program that samples x from the distribution p and returns the *spmf* produced by f when given x. We can also return an *spmf* using the monad operation *return*. The following probabilistic program, *completeness-game*, is used in our formalisation of the correctness property of commitment schemes, given in Section 4. Here *init* and *response* are the probabilistic programs that define the two steps of a Σ-protocol completed by the Prover and *check* is the function that the verifier uses to validate the response. To define the *completeness-game*, *init* and *response* are sampled like in a real execution of a commitment scheme, and the distribution (*spmf*) of *check* is returned. Note, as *check* is deterministic we must return the output as a probability distribution.

$$\begin{aligned}&\mathit{completeness\text{-}game}(h, w, e) = \mathit{do}\ \{\\&\quad (r, a) \leftarrow \mathit{init}(h, w);\\&\quad z \leftarrow \mathit{response}(r, w, e);\\&\quad \mathit{return}(\mathit{check}(h, a, e, z))\}\end{aligned} \tag{2}$$

We note that *bind* is commutative, that is, assuming no dependency conditions one can bind spmfs in any order. In particular, given a sequence of samplings the ordering of such samplings is irrelevant.

Under *bind* we also have that constant elements cancel. In particular if p is lossless (its probability mass sums to one), then

$$\mathit{bind}(p, \lambda\_.\ q) = q. \tag{3}$$

Our 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.

Assertions Making assertions inside probabilistic programs is sometimes useful. For example we must ensure that the adversary in the hiding game (Equation 10) outputs two valid messages for the game to proceed. The monad for subprobabilities has an element, $\bot$, that accounts for failure meaning the current part of the probabilistic program is aborted. This is captured by assertion statements

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

where if b holds then the probabilistic program continues otherwise it fails. Here $(\_)$ is the only element of the unit type, returning this element continues with execution of the program with no effect. Assertions are often used in conjunction with the $\mathit{TRY}\ p\ \mathit{ELSE}\ q$ construct. For example $\mathit{TRY}\ p\ \mathit{ELSE}\ q$ 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's output being a coin flip meaning they do not win the resulting security game.

Assertions are not a necessity to our formalisation as the assumptions could be made explicitly in the theorem statements, for example in any statement of the hiding property we could assume all messages outputted by the adversary ($\mathcal{A}_1$) are valid:

$$\forall vk.\ (m_0, m_1) \in \mathit{set\text{-}spmf}(\mathcal{A}_1) \longrightarrow \mathit{valid\text{-}msg}(m_0) \wedge \mathit{valid\text{-}msg}(m_1).$$

Assertions however, in general, make the formalisation more neat and readable.

Sampling Sampling from sets is important in cryptography. CryptHOL provides an operation *uniform* which returns a uniform distribution over a finite set. We use two cases of this function extensively: by $\mathit{samp\text{-}uniform}(q)$, where q is a natural, we denote the uniform sampling from the set $\{0, \ldots, q-1\}$ and by *coin* we denote the uniform sampling from the set $\{True, False\}$ — a coin flip.

The monad operations give rise to another function, $\mathit{map} :: (\alpha \Rightarrow \beta) \Rightarrow \alpha\ \mathit{spmf} \Rightarrow \beta\ \mathit{spmf}$.

$$\mathit{map}(f, p) = \mathit{bind}(p, (\lambda x.\ \mathit{return}(f(x)))) \tag{4}$$

The map function can be thought of as the *post-processing* of sampled values. It is from this level of abstraction that we are able to reason about the equivalence of distributions and thus complete major steps in our proofs. For example, we can apply one time pad lemmas. Below is that statement of the one time pad for addition in the finite group $\mathbb{Z}_q$.

$$\mathit{map}((\lambda b.\ (y + b)\ \mathit{mod}\ q), (\mathit{samp\text{-}uniform}(q))) = \mathit{samp\text{-}uniform}(q) \tag{5}$$

Probabilities Security definitions are based on explicit probabilities of events occurring. In CryptHOL the expression $\mathcal{P}[Q = x]$ denotes the subprobability mass the spmf Q assigns to the event x. In our proofs reasoning at this level is often the last step, much of the proof effort is in showing properties of the probabilistic programs over which the probabilities are defined.

Negligible functions To reason about security in the asymptotic case we must consider negligible functions. These are formalised as a part of CryptHOL in the canonical way. A function, $f :: \mathit{nat} \Rightarrow \mathit{real}$ is said to be negligible if

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

where o is the little o notation. We discuss the use of such functions in our proofs in Section 9.1.

Cyclic Groups We highlight the formalisation of cyclic groups that CryptHOL provides; the construction provides the user with a cyclic group G and a generator g. The formalisation extends the formalisation of monoids in Isabelle/HOL meaning there is an armoury of lemmas immediately available for use. We use cyclic groups in the formalisation of the Pedersen commitment scheme and the Schnorr, Chaum-Pedersen and Okamoto Σ-protocols. In the formal parts of this paper we denote group multiplication by $\otimes$ whereas we denote the multiplication of natural numbers by $\cdot$. In the informal parts of the paper all multiplication is written as $\cdot$.

3 Formalisation overview

CryptHOL has been used for a number of formalisations of cryptography thus far. Our work lends weight to the fact that CryptHOL provides a good environment for such formalisations, in particular that the method of modularisation can be used for considering low level cryptographic primitives.

In this section we first discuss the general method of our formalisation at a high level, in particular how CryptHOL allows the user to make their definitions abstract and then instantiate them for the proofs we consider. This method could be considered as the general, most effective, method that Isabelle and CryptHOL allow for. Second we briefly discuss asymptotic security statements in CryptHOL.

3.1 Method of formalisation

Isabelle's module system and CryptHOL's monadic structure allow for a natural hierarchy in our 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. The *human reader* needs to only check the high level, abstract, definitions of security to have confidence in the whole collection of proof as all instantiated proofs are made with respect to these definitions. We are able to prove some lemmas at the abstract level and have them at our disposal in any instantiation, thus reducing the workload for future proofs. Some of the properties are technical and uninteresting to the cryptographer, for example we prove losslessness of various probabilistic programs used in the definitions, however we are also able to reason about the properties more generally. For example, to formalise the construction of commitment schemes from Σ-protocols we work at an abstract level, only assuming the existence of a Σ-protocol. This means the instantiated proofs (for the Σ-protocols we consider) come for free once we prove they are Σ-protocols.

We next more explicitly describe the workflow in constructing our formalisation.

3.1.1 Instantiating the abstract frameworks

We use Isabelle's locales to define properties of security relative to fixed constants and then instantiate these definitions for explicit protocols and prove the security properties as theorems.

Below we show how we formalise the completeness property for Σ-protocols. We follow the same process to define and instantiate the other security properties we consider for commitment schemes and Σ-protocols.

1. To consider Σ-protocols abstractly and define correctness we fix in a locale the probabilistic programs (algorithms) that make up the primitive (i.e.

init, *response*, *check*) as well as other parameters of the Σ-protocol — we introduce the other parameters in Section 4.

locale *Σ-protocol-base* =
  fixes *init* :: ('*pub-input* × '*witness*) ⇒ ('*rand* × '*msg*) *spmf*
    and *response* :: '*rand* ⇒ '*witness* ⇒ '*challenge* ⇒ *response spmf*
    and *check* :: '*pub-input* ⇒ '*msg* ⇒ '*challenge* ⇒ '*response* ⇒ *bool*
    and *Rel* :: ('*pub-input* × '*witness*) *set*
    and S_{raw} :: '*pub-input* ⇒ '*challenge* ⇒ ('*msg*, '*response*) *sim-out spmf*
    and $\mathcal{A}_{ss}$ :: ('*pub-input*, '*msg*, '*challenge*, '*response*, '*witness*) *prover-adversary*
    and *challenge-space* :: '*challenge set*
    and *valid-pub* :: '*pub-input set*
  assumes *Domain*(*Rel*) ⊆ *valid-pub*

2. Using these fixed parameters we construct the the probabilistic program *completeness-game*, given in Equation 2 and use it to define the completeness property.

$$completeness = (\forall h\ w\ e.\ (h,w) \in Rel \longrightarrow e \in challenge\text{-}space \longrightarrow \mathcal{P}[completeness\text{-}game(h,w,e) = True] = 1)$$

   Here we say the Σ-protocol is complete if for all valid challenges the completeness game returns true.
3. To instantiate a Σ-protocol and prove it is complete we explicitly define the fixed parameters from the locale, *Σ-protocol-base*. To do this we refine the types and define the probabilistic programs that describe the protocol. In the case of the Schnorr Σ-protocol we work with a cyclic group G by fixing it in the locale *schnorr-base*.

   locale *schnorr-base* =
     fixes G :: '*grp cyclic-group*
     assumes *prime*($|G|$)

   Inside this locale we define the instantiated parameters: $init^S, response^S, check^S, Rel^S, S_{raw}, \mathcal{A}_{ss}, challenge\text{-}space^S$ and $valid\text{-}pub^S$ — here the superscript S denotes they are the parameters for the Schnorr protocol.
4. We then utilise Isabelle's locale structure by importing the abstract theory using the **sublocale** command.

$$\textbf{sublocale}\ schnorr\text{-}\Sigma :\ \Sigma\text{-}protocol\text{-}base\ init^S\ response^S\ check^S\ Rel^S\ S_{raw}\ \mathcal{A}_{ss}\ challenge\text{-}space^S\ valid\text{-}pub^S \quad (6)$$

   Not only must the explicit definitions be of the correct type when importing a locale, one must also discharge any assumptions that come with the locale. This means that our instantiation is valid with respect to the *Σ-protocol-base* locale and we can refer its definition of correctness. In this case we must prove that $Domain(Rel^S) \subseteq valid\text{-}pub^S$.

5. Any call of a definition from the original locale (in this case *Σ-protocol-base*) requires the definition name to be prefixed by the name we give to the sublocale (in this case *Schnorr-Σ*). The statement of completeness for the Schnorr Σ-protocol is now given by *schnorr-Σ.completeness*.

3.2 Concrete vs. asymptotic security

In our formalisation, we first prove *concrete* security bounds using reduction-style proofs. That is, we bound on adversary's advantage as a function of advantages of different adversaries of the primitives used in the construction. For example, we show in Lemma 10 in Section 8.2 that the binding advantage for commitment schemes constructed from Σ-protocols is bounded by the advantage that the (transformed) adversary breaks the hard relation *Rel*. This is in line with other CryptHOL formalisations [12, 30].

From these concrete statements, we can easily derive more abstract asymptotic security statements. To that end, a security parameter must be introduced. We describe in Section 9.1 how we achieve this with little effort using Isabelle's locale system. Conceptually, this process replaces a locale parameter such as the cyclic group $\mathcal{G}$ :: *'grp cyclic-group* with a family of cyclic groups $\mathcal{G}$ :: *nat* ⇒ *'grp cyclic-group*. And similarly, the challenge space *challenge-space* becomes a family of type *nat* ⇒ *'challenge set*. This parameterisation is also the reason for the locale parameters *valid-pub* and *challenge-space*. Since HOL does not have dependent types, the same abstract type *'challenge* must hold the challenge spaces for every possible security parameter value. The parameter *challenge-space* then carves out the right challenge space for the chosen security parameter.

Unfortunately, CryptHOL cannot reason about computational aspects, due to the shallow embedding. We therefore cannot formalise notions like computational binding (Definition 6) that quantify over computationally bounded adversaries. Instead, we capture the underlying reduction argument in a reduction-based security theorem. As an example, for constructing a commitment scheme from a Σ-protocol, the concrete security theorem has the following form: the binding advantage *bind-adv*($\mathcal{A}$) of an adversary $\mathcal{A}$ is bounded by the advantage of a different adversary $\mathcal{A}'$ against the hardness of the underlying relation *Rel*. This adversary $\mathcal{A}'$ is obtained by a reduction f, which systematically transforms binding-game adversaries $\mathcal{A}$ into hardness game adversaries $\mathcal{A}' = f(\mathcal{A})$. Such statements naturally yield asymptotic security statements of the following form: The binding advantage of a family of adversaries $\mathcal{A}_\eta$ against the commitment scheme is negligible if the family of reduced adversaries $f(\mathcal{A}_\eta)$ has negligible advantage against the hardness of the underlying relation.

Such a reduction-based statement captures the key aspects of the security proof. Compared to a computational statement, which quantifies over all computationally bounded adversaries, the reduction f shows up in the security statement itself. This makes the statement more generic in the sense that we need not commit to a particular computational model or complexity class such as polynomial time. Conversely, the reader must manually check that the reduction lies in the desired complexity class.

4 Formalising Σ-Protocols

In this section we detail our formalisation of Σ-protocols based on the definitions from Section 2.1.1.

As explained in the previous section, we define a locale where we fix the parameters required for the definitions (Figure 4). That is we fix, as probabilistic programs, the components of the Σ-protocol:

- *init* constructs the initial message sent from P to V, and its corresponding randomness.
- *response* is the response sent from P to V.
- *check* performs the verification V runs on the response from P.

We also fix the relation *Rel*, the adversary $\mathcal{A}_{ss}$ required in the special soundness definition, the *challenge-space* which is the set of all possible challenges and the set *valid-pub* which contains all the valid public inputs. We also require a simulator for the HVZK definition: the simulator outputs a conversation of the form (a, e, z), however the outputted challenge e must be the same as the inputted challenge e; overall the simulator looks as follows:

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

To formally model this we fix in the locale the part of the simulator, S_{raw}, that constructs a and z and then define the full simulator that outputs (a, e, z) using S_{raw} as follows:

$$S(h, e) = map(\lambda\ (a, z).\ (a, e, z), S_{raw}(h, e)).$$

To improve the readability of the formalisation we define three type synonyms; the first two define the type of S_{raw} and a conversation respectively and the third the type of the special soundness adversary.

type-synonym $('msg, 'response)\ \textit{sim-out} = ('msg \times 'response)$

type-synonym $('msg, 'challenge, 'response)\ \textit{conv-tuple} =$
$('msg \times 'challenge \times 'response)$

type-synonym
$('pub\text{-}input, 'msg, 'challenge, 'response, 'witness)\ \textit{prover-adversary}$
$= 'pub\text{-}input \Rightarrow ('msg, 'challenge, 'response)\ \textit{conv-tuple}$
$\Rightarrow ('msg, 'challenge, 'response)\ \textit{conv-tuple} \Rightarrow 'witness\ spmf$

```
locale $\Sigma$-protocol-base =
  fixes $init$ :: ('pub-input × 'witness) ⇒ ('rand × 'msg) spmf
    and $response$ :: 'rand ⇒ 'witness ⇒ 'challenge ⇒ response spmf
    and $check$ :: 'pub-input ⇒ 'msg ⇒ 'challenge ⇒ 'response ⇒ bool
    and $Rel$ :: ('pub-input × 'witness) set
    and $S_{raw}$ :: 'pub-input ⇒ 'challenge ⇒ ('msg, 'response) sim-out spmf
    and $\mathcal{A}_{ss}$ :: ('pub-input, 'msg, 'challenge, 'response, 'witness) prover-adversary
    and $challenge$-$space$ :: 'challenge set
    and $valid$-$pub$ :: 'pub-input set
  assumes $Domain(Rel) \subseteq valid$-$pub$
```

Fig. 4: Locale fixing the constants for *Σ*-protocols.

The locale where we fix these parameters is given in Figure. 4 — note this is the same as the locale given in the running example in Section 3. The assumption requires that the domain of the relation is contained in the set of valid public inputs. We now make our formalised definitions of *Σ*-protocols.

The set L is the set of all public inputs for which a witness exists such that the relation holds.

$$L = \{x.\ \exists w.\ Rel(x, w)\}$$

Using the parameters we fixed in the locale *Σ-protocol-base* we define the properties of *Σ*-protocols. First we define completeness. For this property we define a probabilistic program, *completeness-game*, that runs the components of the protocol and outputs the output of *check*. We repeat the definition from Equation 2.

$$\begin{aligned} & completeness\text{-}game(h, w, e) = do\ \{ \\ & \quad (r, a) \leftarrow init; \\ & \quad z \leftarrow response(r, w, e); \\ & \quad return(check(h, a, e, z))\} \end{aligned} \tag{7}$$

The definition of completeness is quantified over all public inputs, witnesses and challenges.

Definition 1

$$\begin{aligned} completeness = (\forall h\ w\ e.\ (h, w) \in Rel \longrightarrow e \in challenge\text{-}space \\ \longrightarrow \mathcal{P}[completeness\text{-}game(h, w, e) = True] = 1) \end{aligned}$$

For special soundness to hold we require the special soundness adversary ($\mathcal{A}_{ss}$) to output the witness when given two accepting conversations (with distinct challenges) with respect the public input h, (a, e, z) and (a, e', z'). An accepting conversation is a tuple upon which *check* is satisfied. To capture this formally we must show that for all w' in the support set (*set-spmf*) of $\mathcal{A}_{ss}$ the relation is satisfied. Together with this we require that $\mathcal{A}_{ss}$ is lossless, if not $\mathcal{A}_{ss}$ may output nothing leaving no way to reason about all outputs of $\mathcal{A}_{ss}$.

Definition 2

$$\begin{aligned}
&\textit{special-soundness} = (\forall h\ a\ e\ z\ e'\ z'.\ h \in \textit{valid-pub}\\
&\qquad \longrightarrow e \in \textit{challenge-space} \longrightarrow e' \in \textit{challenge-space} \longrightarrow e \neq e'\\
&\qquad\qquad \longrightarrow \textit{check}(h, a, e, z) \longrightarrow \textit{check}(h, a, e', z') \longrightarrow\\
&\qquad\qquad\quad \textit{lossless}(\mathcal{A}_{ss}(h, (a, e, z), (a, e', z'))) \wedge\\
&\qquad\qquad\quad \forall w' \in \textit{set-spmf}(\mathcal{A}_{ss}(h, (a, e, z), (a, e', z'))).\ \textit{Rel}(h, w'))
\end{aligned}$$

The definition of HVZK follows the simulation-based paradigm: we require the output distribution of the simulator S to be equal to the output distribution of the real view of the protocol which is given below.

$$\begin{aligned}
&\textit{real-view}(h, w, e) = \textit{do}\ \{\\
&\quad (r, a) \leftarrow \textit{init};\\
&\quad z \leftarrow \textit{response}(r, w, e);\\
&\quad \textit{return}(a, c, z)\}
\end{aligned}$$

The real view can be defined abstractly as we know the structure of the protocol. This is unlike in general MPC protocols [12] where the real view has to be defined for each MPC protocol considered. We must nevertheless construct a simulator for each instantiated Σ-protocol. As noted in Section 2.1.1, we additionally require that the simulator's output produces an accepting conversation even if the public input h does not belong to the language.

Definition 3

$$\begin{aligned}
&\textit{HVZK} = (\forall e \in \textit{challenge-space}.\\
&\qquad\qquad (\forall (h, w) \in \textit{Rel}.\ \textit{real-view}(h, w, e) = S(h, e))\\
&\qquad \wedge (\forall h \in \textit{valid-pub}.\ \forall (a, z) \in \textit{set-spmf}(S_{raw}(h, e)).\ \textit{check}(h, a, e, z)))
\end{aligned}$$

Using these three definitions we define the notion of a Σ-protocol.

Definition 4 (Σ-protocol)

$$\Sigma\textit{-protocol} = \textit{completeness} \wedge \textit{special-soundness} \wedge \textit{HVZK}$$

It may appear surprising that in our formalisation of Σ-protocols we do not fix a probabilistic program to output the challenge, like we do for the other components of the protocol. In this case it is not needed as the verifier, who outputs the challenge, is assumed to be honest. In particular we define the properties over all allowed challenges ($\forall e \in \textit{challenge-space}$). This is valid when the challenge is always generated honestly, however is not strong enough if we moved to assume the challenge was not generated honestly — in the case of a corrupt verifier. This extension is considered by full Zero-Knowledge protocols, which we do not consider in this work.

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

Fig. 5: The Schnorr Σ-protocol.

5 The Schnorr Σ-protocol

In this section we detail how we instantiate our formal definitions of Σ-protocols given in Section 4 for the Schnorr Σ-protocol. We first explain the protocol in Section 5.1 and give some intuition and informal arguments as to why the desired properties hold and then in Section 5.2 we detail our formalisation.

5.1 The Schnorr Σ-protocol

The Schnorr protocol uses a cyclic group G with generator g and considers the discrete log relation which on public input h requires the witness to be the discrete log of h in G — $h = g^w$. The Schnorr Σ-protocol is given in Figure 5.

The Prover holds (h, w) such that $h = g^w$ and the Verifier holds only h. The initial message sent by P to V is a uniformly sampled group element and the challenge is uniformly sampled from the field of size $|G|$. The response is constructed by P as $z = (w \cdot e + r)\ mod\ |G|$ and sent to V who accepts or rejects based on whether $a \cdot h^e = g^z$.

Completeness comes directly by unfolding the definitions and proving the identity $g^r \cdot (g^w)^e = g^{r+w \cdot e}$.

For the special soundness property a witness can be extracted from two accepting conversations (a, e, z) and (a, e', z') by taking $w = (\frac{z-z'}{e-e'})\ mod\ |G|$. This can be seen as follows. Given two accepting conversations (a, e, z) and (a, e', z') we have $a \cdot h^e = g^z$ and $a \cdot h^{e'} = g^{z'}$ which after unfolding $h = g^w$ and rearranging leaves us with $g^{z-w \cdot e} = g^{z'-w \cdot e'}$ meaning we have $[z - w \cdot e = z' - w \cdot e']\ mod\ |G|$. Rearranging this we find $w = (\frac{z-z'}{e-e'})\ mod\ |G|$ as claimed. Note it is important that $[e \neq e']\ mod\ |G|$, this comes from $e, e' < |G|$ (the challenges are from $\mathbb{Z}_{|G|}$) and $e \neq e'$ (a condition on the special soundness property).

The protocol also observes the HVZK property. The intuition behind constructing the simulator for the HVZK property is to work backwards. We would like the response to leak no information about w, so let us pick it uniformly at random and then try to reconstruct the initial message. If we sample z uniformly from the field and then set $a = g^z \cdot h^{-e}$ it can be shown the resulting conversation gives a distribution equal to the output conversation distribution of a real execution of the protocol.

5.2 Formalising the Schnorr Σ-protocol

Throughout our formalisation we work with natural numbers instead of formalising a field construction. Therefore we work modulo q whenever we actually work in a field. One issue we encounter is constructing inverses modulo q. We are required to reason about the inverses of elements in a field in many places in our formalisation, for example the special soundness adversary outputs $w = (\frac{z-z'}{e-e'})\ mod\ |G|$ in the Schnorr protocol. Before we detail our formalisation of the Schnorr Σ-protocol we show how we formalise such an inverse.

Obviously, the standard division function on natural numbers is not suitable to obtain an inverse in the field modulo q. Instead, we use the existing number theory formalisation in Isabelle's standard library, in particular Bezout's function (*bezw*). Bezout's identity informally says: let a and b be integers such that $gcd(a, b) = d$ then there exist integers x and y such that $a \cdot x + b \cdot y = d$. In Isabelle, the function $bezw(a, b)$ returns the pair (x, y) of witnesses to Bezout's identity. So we obtain the inverse of a as $fst(bezw(a, q))$. For readability we define an abbreviation for the inverse.

$$inv_q(a) = fst(bezw(a, q))$$

We prove the following general lemma, which we find is sufficient in all the cases where reasoning about the inverse is required in our formalisation.

Lemma 1 **assumes** $gcd(a, q) = 1$
shows $[a \cdot inv_q(a) = 1]\ mod\ q$

Proof The function *bezw* outputs a pair of witnesses to Bezout's identity, using this along with the assumption that $gcd(a, q) = 1$ we have

$$inv_q(a) \cdot a + snd(bezw(a, q)) \cdot q = 1$$

Considering this modulo q the result comes easily as the second term on the left hand side vanishes. $\square$

In the case of the Schnorr Σ-protocol we instantiate q as $|G|$. The assumption, in general, holds in our usage as $a < |G|$, $a \neq 0$ and $|G|$ is prime.

The Schnorr Σ-protocol is defined over a cyclic group of prime order. We use the construction of cyclic groups from [31] to fix a group $\mathcal{G}$ in the locale we work in as follows.

locale *schnorr-base* =
  fixes $\mathcal{G}$ :: *'grp cyclic-group* (**structure**)
  assumes $prime(order(\mathcal{G}))$

To show the Schnorr Σ-protocol has the desired properties of Σ-protocols we explicitly define the constants introduced in Section 4. We define

$$\mathit{init}^S, \mathit{response}^S, \mathit{check}^S, R^S_{DL}, S^S_{raw}, \mathcal{A}^S_{ss}, \mathit{challenge\text{-}space}^S, \mathit{valid\text{-}pub}^S$$

where the superscript S denotes that these constants are for the Schnorr Σ-protocol. We make these definitions inside the context of the locale. The types of the components of the protocol are made more concrete from definitional theory of Σ-protocols, in particular we define the following type synonyms.

type-synonym *witness* = *nat*
type-synonym *'grp pub-in* = *'grp*
type-synonym *'grp msg* = *'grp*
type-synonym *rand* = *nat*
type-synonym *challenge* = *nat*
type-synonym *response* = *nat*

These new types specialize the types from the definitional theory to the Schnorr protocol. For example, the witness, randomness, challenge and response are all naturals and the public input and initial message are group elements.

For the Schnorr Σ-protocol the relation is the discrete log relation, as given informally in Equation 1; formally this is encoded into Isabelle as

$$R^S_{DL} = \{(h, w).\ h = g^w\}.$$

The programs $\mathit{init}^S, \mathit{response}^S$ and check^S correspond to the stages of the protocol given in Figure 5.

init^S :: $(\text{'}\mathit{grp\ pub\text{-}in} \times \mathit{witness}) \Rightarrow (\mathit{rand} \times \text{'}\mathit{grp\ msg})\ \mathit{spmf}$
$\mathit{init}^S(h, w) = \mathit{do}\ \{$
  $r \leftarrow \mathit{samp\text{-}uniform}(|G|);$
  $\mathit{return}(r, g^r)\}$

$\mathit{response}^S$ :: $\mathit{rand} \Rightarrow \mathit{witness} \Rightarrow \mathit{challenge} \Rightarrow \mathit{response}\ \mathit{spmf}$
$\mathit{response}^S(r, w, e) = \mathit{return}((w \cdot c + r)\ \mathit{mod}\ |G|)$

check^S :: $\text{'}\mathit{grp\ pub\text{-}in} \Rightarrow \text{'}\mathit{grp\ msg} \Rightarrow \mathit{challenge} \Rightarrow \mathit{response} \Rightarrow \mathit{bool}$
$\mathit{check}^S(h, a, e, z) = (a \otimes h^e = g^z)$

A public input is valid if it is in the group, $\mathit{valid\text{-}pub}^S = \mathit{carrier}(G)$. And the challenge set is the set of naturals up to the order of G, $\mathit{challenge\text{-}space}^S = \{0, \ldots, |G|\}$.

We show these constants are an instantiation of the *Σ-protocol-base* locale (Figure 4). As explained in Section 3.1.1 we do this using the sublocale command; this is an extension of the sublocale given in Equation 6.

sublocale *Schnorr-Σ* : *Σ-protocol-base* init^S $\mathit{response}^S$ check^S
R^S_{DL} S^S_{raw} $\mathcal{A}^S_{ss}$ $\mathit{challenge\text{-}space}^S$ $\mathit{valid\text{-}pub}^S$

We also inherit the cyclic group properties of the group G by forming the following locale.

locale *schnorr* = *schnorr-base* + *cyclic-group*(G)

In this context we can prove the desired properties of the Schnorr Σ-protocol.

Lemma 2 *(in schnorr)* **shows** *Schnorr-Σ.completeness*

Proof Completeness follows after proving the identity $g^r \otimes (g^w)^e = g^{r+w \cdot e}$ and passing it as a rewrite rule to the simplifier. □

Second we consider special soundness. To prove this property we construct an adversary that can extract the witness from accepting conversations of the protocol. We informally gave the construction of this adversary in the previous section; given two accepting conversations (a, e, z) and (a, e', z') the adversary outputs $(\frac{z-z'}{e-e'})\ mod\ |G|$. The encoding of the adversary in Isabelle must be mindful of whether $e > e'$; as we are working with naturals bounded subtraction in the denominator $e - e'$ will return 0 if $e < e'$. So we construct an adversary that is mindful of this — we know that $e \neq e'$ as it is a condition on the conversations given to the adversary.

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

Using this adversary we prove the special soundness property for the Schnorr Σ-protocol.

Lemma 3 *(in schnorr)* **shows** *Schnorr-Σ.special-soundness*

Proof The adversary $\mathcal{A}_{ss}^S$ is clearly lossless — it does not do any probabilistic sampling. Showing the adversary outputs a witness to the relation is proven by using Lemma 1 to rewrite the output of the adversary in a similar manner to a paper proof given in Section 5.1. □

Finally we consider the honest verifier zero knowledge property. This proof technique follows the technique of simulation-based proofs that was formally introduced in Isabelle and CryptHOL in [12]. To prove HVZK we define the simulator, S_{raw}^S, which in turn defines *Schnorr-Σ.S^S*. We then prove this mimicks the real view. The unfolded simulator is formed as follows; recall the intuition of sampling the response first and constructing the initial message from it.

```
$Schnorr\text{-}\Sigma.S^S(h, e) = do\ \{$
  $z \leftarrow samp\text{-}uniform(|G|);$
  $let\ a = g^z \otimes (h^e)^{-1};$
  $return\ (a, e, z)\}$
```

Lemma 4 *(in schnorr)* **shows** *Schnorr-Σ.HVZK*(h, w)

Proof First we show the simulator and the real view are equal. The unfolded real view can be written as:

```
$Schnorr\text{-}\Sigma.real\text{-}view^S(h, w) = do\ \{$
  $r \leftarrow samp\text{-}uniform(|G|);$
  $let\ (r, a) = (r, g^r);$
  $c \leftarrow samp\text{-}uniform(|G|);$
  $let\ z = (w \cdot c + r)\ mod\ |G|;$
  $return\ (a, c, z)\}$
```

The juxt of the proof is showing that z constructed in the real view is a uniform sample — as it is in the simulator — this destroys any information passed to V about the witness. To do this we use the following one time pad lemma:

$$map(\lambda b.\ (y + b)\ mod\ q,\ samp\text{-}uniform(q)) = samp\text{-}uniform(q)$$

To use this lemma in the proof we must rewrite some of the terms in the real view. These rewriting statements of equality are nearly always needed when using such lemmas as the remaining probabilistic program can no longer depend on b and must be rewritten in terms of the other variables in the probabilistic program.

Second we show the output of the simulator is a valid transcript. This part of the proof comes easily and in a similar manner to the proof of correctness. □

Using Lemmas 2, 3 and 4 we show that the Schnorr *Σ*-protocol satisfies the definition of a *Σ*-protocol given in Section 4.

Theorem 1 *(in schnorr)* **shows** *Schnorr-Σ.Σ-protocol*

6 Compound *Σ*-protocols

Σ-protocols can be combined to prove knowledge for AND and OR statements. Consider two *Σ*-protocols, Σ_0 and Σ_1, with relations Rel_0 and Rel_1 respectively. The AND construction allows the prover to prove they know witnesses w_0 and w_1 such that both $Rel_0(x_0, w_0)$ and $Rel_1(x_1, w_1)$ are true and the OR construction allows for the proof of knowledge of a witness such that $Rel_0(x_0, w)$ or $Rel_1(x_1, w)$ is true — (x_0, x_1) is the public input. Cryptographers have found many uses for these basic constructions, for example the voting protocols in [20]. In this section we detail our formalisation of the OR construction, details of the AND construction can be found in Appendix C.

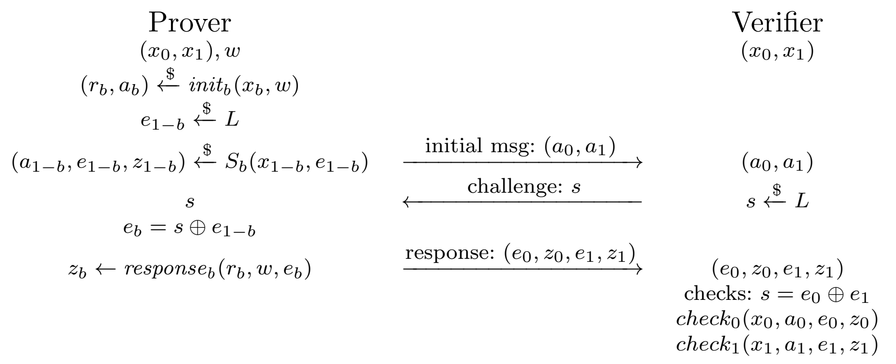
Prover
$(x_0, x_1), w$
$(r_b, a_b) \overset{\$}{\leftarrow} init_b(x_b, w)$
$e_{1-b} \overset{\$}{\leftarrow} L$
$(a_{1-b}, e_{1-b}, z_{1-b}) \overset{\$}{\leftarrow} S_b(x_{1-b}, e_{1-b})$
s
$e_b = s \oplus e_{1-b}$
$z_b \leftarrow response_b(r_b, w, e_b)$
initial msg: (a_0, a_1)
challenge: s
response: (e_0, z_0, e_1, z_1)
Verifier
(x_0, x_1)
(a_0, a_1)
$s \overset{\$}{\leftarrow} L$
(e_0, z_0, e_1, z_1)
checks: $s = e_0 \oplus e_1$
$check_0(x_0, a_0, e_0, z_0)$
$check_1(x_1, a_1, e_1, z_1)$

Fig. 6: The OR construction for two Σ-protocols, Σ_0 and Σ_1. L is the boolean algebra that the protocol is run over. (x_0, x_1) is the public input such that $Rel_0(x_0, w)$ or $Rel_1(x_1, w)$ is satisfied and b represents the relation that holds, that is we have that $Rel_b(x_b, w)$.

6.1 The OR construction

The construction of the OR protocol follows the idea that the prover can run the real protocol for the relation for which the witness is known and run the simulator to generate the conversation for the relation for which the witness is not known. By the HVZK property of Σ-protocols the simulated view is equivalent to the real view, therefore the verifier cannot tell which was constructed by the real protocol and which from the simulator. The protocol is shown in Figure 6. In this section we just give the statement of the lemmas, the proofs can be found in Appendix A.

In the literature [20, 22, 29] the OR construction is considered over bit-strings. However we only require the one time pad property of the xor function thus we are able to generalise the construction to arbitrary boolean algebras. To do this we formalise the concept of a boolean algebra and prove the one time pad property, whose statement is seen in Equation 8.

$$map((\lambda a.\ a \oplus x), (uniform(carrier(L))) = uniform(carrier(L)) \qquad (8)$$

where L is the boolean algebra with xor function $\oplus$.

To formalise the OR construction we fix two Σ-protocols (Σ_0 and Σ_1) and their respective components

$$init_i, response_i, check_i, Rel_i, S_{raw,i}, \mathcal{A}_{ss,i}, challenge\text{-}space_i, valid\text{-}pub_i$$

for $i \in \{0, 1\}$ as well as a boolean algebra L :: '*bool-alg boolean-algebra*. The only type constraint on the components of Σ_0 and Σ_1 is that both challenges must be of type '*bool-alg*. We allow the types of Σ_0 and Σ_1 to be different, thus the witness must be a sum type w :: ('$witness_0$ + '$witness_1$).

We define the relation,

$$Rel_{OR} :: (('pub_0 \times 'pub_1) \times ('witness_0 + 'witness_1))\ set$$

as an inductive set with the following introduction rules:

$$((x_0, x_1), \mathit{Inl}(w_0)) \in \mathit{Rel}_{OR} \textbf{ if } (x_0, w_0) \in \mathit{Rel}_0 \wedge x_1 \in \mathit{valid\text{-}pub}_1$$
$$((x_0, x_1), \mathit{Inr}(w_1)) \in \mathit{Rel}_{OR} \textbf{ if } (x_1, w_1) \in \mathit{Rel}_1 \wedge x_1 \in \mathit{valid\text{-}pub}_0$$

In particular the prover knows a witness for one of the two relations, and knows to which relation the witness belongs to. We also require that the public input for which the prover does not know the witness is a valid public input for its respective Σ-protocol.

In the OR construction the initial message is constructed as either the real initial message (of the Σ-protocol for which the prover knows the witness) or the first message of the simulator (of the other Σ-protocol). init_{OR}'s output has two parts: 1. the randomness consisting of the randomness from init_b (where $b \in \{0, 1\}$ is the relation for which the prover knows the witness), the random challenge sampled, as well as the response from the conversation that is simulated and 2. the initial messages sent in the protocol, one (and only one) of which is constructed by the simulator.

```
$\mathit{init}_{OR}((x_0, x_1), \mathit{Inl}(w_0)) = \mathit{do}$ {
  $(r_0, a_0) \leftarrow \mathit{init}_0(x_0, w_0)$;
  $e_1 \leftarrow \mathit{uniform}(\mathit{carrier}(L))$;
  $(a_1, e_1, z_1) \leftarrow S_1(x_1, e_1)$;
  $\mathit{return}(\mathit{Inl}(r_0, e_1, z_1), (a_0, a_1))$}
$\mathit{init}_{OR}((x_0, x_1), \mathit{Inr}(w_1)) = \mathit{do}$ {
  $(r_1, a_1) \leftarrow \mathit{init}_1(x_1, w_1)$;
  $e_0 \leftarrow \mathit{uniform}(\mathit{carrier}(L))$;
  $(a_0, e_0, z_0) \leftarrow S_0(x_0, e_0)$;
  $\mathit{return}(\mathit{Inr}(r_1, e_0, z_0), (a_0, a_1))$}
```

To respond to a challenge, s, the prover constructs a new challenge to be used in constructing the real response by xoring it with the challenge e it generated in init_{OR}. The response for the relation the prover does not know is given as the simulated response from the init_{OR} phase. The inputs to $\mathit{response}_{OR}$ consist of 1. the randomness outputted by init_{OR} (a 3-tuple) 2. the witness that is known and 3. the challenge.[5]

```
$\mathit{response}_{OR}(\mathit{Inl}(r_0, e_1, z_1), \mathit{Inl}(w_0), s) = \mathit{do}$ {
  $\mathit{let}\ e_0 = s \oplus e_1$;
  $z_0 \leftarrow \mathit{response}_0(r_0, w_0, e_0)$;
  $\mathit{return}((e_0, z_0), (e_1, z_1))$}
$\mathit{response}_{OR}(\mathit{Inr}(r_1, e_0, z_0), \mathit{Inr}(w_1), s) = \mathit{do}$ {
  $\mathit{let}\ e_1 = s \oplus e_0$;
  $z_0 \leftarrow \mathit{response}_1(r_1, w_1, e_1)$;
  $\mathit{return}((e_0, z_0), (e_1, z_1))$}
```

To check the responses given by the prover, the verifier checks both conversations it receives are valid with respect the Σ-protocols they correspond

[5] In this section we denote the challenge as s to distinguish it from the challenges of the underlying Σ-protocols which we will denote with e_0 and e_1.

to as well as checking that the challenge they provided, s, is the xor of the challenges in the respective conversations — $s = e_0 \oplus e_1$.

$$\begin{aligned}
&\mathit{check}_{OR}((x_0,x_1),(a_0,a_1),s,((e_0,z_0),(e_1,z_1)))\\
&\qquad = (s = e_0 \oplus e_1 \wedge e_0 \in \mathit{challenge\text{-}space} \wedge e_1 \in \mathit{challenge\text{-}space}\\
&\qquad\qquad\qquad\qquad \wedge \mathit{check}_0(x_0,a_0,e_0,z_0) \wedge\ \mathit{check}_1(x_1,a_1,e_1,z_1))
\end{aligned}$$

The *challenge-space* is defined as the carrier set of L — $\mathit{challenge\text{-}space}_{OR} = \mathit{carrier}(L)$ and the public input (x_0,x_1) is valid if x_i is a valid public input with respect to its underlying Σ-protocol, that is:

$$\mathit{valid\text{-}pub}_{OR} = \{(x_0,x_1).\ x_0 \in \mathit{valid\text{-}pub}_0 \wedge x_1 \in \mathit{valid\text{-}pub}_1\}.$$

As usual we import the *Σ-protocol-base* locale — this time under the name *Σ-OR* — so we can reason about the properties of Σ-protocols. First we show completeness.

The proof of the completeness property requires Condition 2 of the HVZK definition in Definition 2. It is required because the simulated transcript in the OR protocol must also produce a valid conversation if the verifier is to accept the proof, without Condition 2 we have no guarantee that this is the case.

Lemma 5 *(in Σ-OR-proof)* **shows** *Σ-OR.completeness*

To prove HVZK we use the following simulator, as always this is constructed by defining $S_{raw,OR}$.

$$\begin{aligned}
&\Sigma\text{-}OR.S_{OR}((x_0,x_1),s) = do\ \{\\
&\quad e_1 \leftarrow \mathit{uniform}(\mathit{carrier}(L));\\
&\quad (a_1,e_1',z_1) \leftarrow S_1(x_1,e_1);\\
&\quad \mathit{let}\ e_0 = s \oplus e_1;\\
&\quad (a_0,e_0',z_0) \leftarrow S_0(x_0,e_0);\\
&\quad \mathit{let}\ z = ((e_0',z_0),(e_1',z_1));\\
&\quad \mathit{return}((a_0,a_1),s,z)\}
\end{aligned} \tag{9}$$

Note, in constructing the simulator we had a design choice: sample either e_1 or e_0 and constructing the other — either choice results in the same simulator, this can be seen by applying Equation 8.

Lemma 6 *(in Σ-OR-proof)* **shows** *Σ-OR.HVZK*

To construct the special soundness adversary we condition on the case $e_0 \neq e_0'$. The reason for this is that in the proof of the special soundness property we sohw that either $e_0 \neq e_0'$ or $e_1 \neq e_1'$ must hold (depending on which relation to witness pertains to). In either case the adversary outputs the witness to the respective relation using the special soundness adversaries from Σ_0 or Σ_1.

```
**locale** commit-base =
  **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
```

Fig. 7: Abstract commitment scheme locale.

```
$\mathcal{A}_{ss,OR}((x_0, x_1), conv, conv') = do\ \{$
  $let\ ((a_0, a_1), s, (e_0, z_0), e_1, z_1) = conv;$
  $let\ ((a_0, a_1), s', (e'_0, z'_0), e'_1, z'_1) = conv';$
  $if\ (e_0 \neq e'_0)\ then\ do\ \{$
    $w_0 \leftarrow \mathcal{A}_{ss,0}(x_0, (a_0, e_0, z_0), (a_0, e'_0, z'_0));$
    $return(Inl(w_0))\}$
  $else\ do\{$
    $w_1 \leftarrow \mathcal{A}_{ss,1}(x_1, (a_1, e_1, z_1), (a_1, e'_1, z'_1));$
    $return(Inr(w_1))\}\ \}$
```

Lemma 7 *(in Σ-OR-proof)* **shows** *Σ-OR.special-soundness*

Using Lemmas 5, 6 and 7 we can prove the OR construction is a *Σ*-protocol.

Theorem 2 *(in Σ-OR-proof)* **shows** *OR-Σ.Σ-protocol*

7 Formalising Commitment Schemes

We formalise commitment schemes analogously to *Σ*-protocols. First we fix the required parameters in the locale, *commit-base*, given in Figure 7.

The probabilistic programs *key-gen*, *commit* and *verify* correspond to the three components of a commitment scheme. The key generation algorithm outputs the keys that are available to the committer and verifier. If, for example, all the keys are public then we have $ck = vk$. The predicate *valid-msg* ensures the messages outputted by the adversary in the hiding game are valid, for example we may require them to be group elements.

Using these fixed parameters we define the correctness, hiding and binding for commitment schemes.

For the correctness property we define the probabilistic program *correct-game*.

```
$correct\text{-}game(m) = do\ \{$
  $(ck, vk) \leftarrow key\text{-}gen;$
  $(c, d) \leftarrow commit(ck, m);$
  $return(verify(vk, m, c, d))\}$
```

For a commitment scheme to be correct we require that for all valid messages *correct-game* always returns True.

Definition 5

$$correct = (\forall m.\ valid\text{-}msg(m) \longrightarrow \mathcal{P}[correct\text{-}game(m) = True] = 1)$$

When considering the hiding and binding properties we define the advantage an adversary has of winning the corresponding security game as well as perfect hiding and binding.

The hiding game, *hiding-game* is defined as follows.

$$\begin{aligned}
&\textit{hiding-game}\ (\mathcal{A}_1, \mathcal{A}_2) = \textit{TRY do}\ \{\\
&\quad (ck, vk) \leftarrow \textit{key-gen};\\
&\quad ((m_0, m_1), \sigma) \leftarrow \mathcal{A}_1(vk);\\
&\quad \_ \leftarrow \textit{assert}(\textit{valid-msg}(m_0) \wedge \textit{valid-msg}(m_1));\\
&\quad b \leftarrow \textit{coin};\\
&\quad (c, d) \leftarrow \textit{commit}(ck, (\textit{if}\ b\ \textit{then}\ m_1\ \textit{else}\ m_2));\\
&\quad b' \leftarrow \mathcal{A}_2(c, \sigma);\\
&\quad \textit{return}(b = b')\}\ \textit{ELSE coin}
\end{aligned} \tag{10}$$

In this game the challenger asks the adversary to output two messages, commits one of the messages and hands it back to the adversary who must determine which message was committed. The adversary is said to win the game if it guesses correctly. Formally the adversary is split into two parts $(\mathcal{A}_1, \mathcal{A}_2)$, the first part outputs the messages and the second its guess at which messages was committed to. We highlight that we must check the messages (m_0, m_1) outputted by the adversary are valid, if the assertion fails then the *ELSE* branch is invoked and the adversary only wins the game half the time (equivalent to if it guessed randomly). Also note the two parts of the adversary must be allowed to pass state to each other. The hiding advantage is defined with respect to the hiding game.

Definition 6 $\textit{hiding-advantage}(\mathcal{A}) = |\mathcal{P}[\textit{hiding-game}(\mathcal{A}) = \textit{True}] - \frac{1}{2}|$

Definition 7 $\textit{perfect-hiding}(\mathcal{A}) = (\textit{hiding-advantage}(\mathcal{A}) = 0)$

The binding game asks the adversary to output a commitment c and two pairs of messages and opening values $((m, d), (m', d'))$ such that they both verify — the messages outputted by the adversary must be distinct and valid, with respect to c, which is accounted for by the assert statement.

$$\begin{aligned}
&\textit{binding-game}\ \mathcal{A} = \ \textit{TRY do}\ \{\\
&\quad (ck, vk) \leftarrow \textit{key-gen};\\
&\quad (c, m, d, m', d') \leftarrow \mathcal{A}(ck);\\
&\quad \_ \leftarrow \textit{assert}(m \neq m' \wedge \textit{valid-msg}(m) \wedge \textit{valid-msg}(m'));\\
&\quad b \leftarrow \textit{verify}(vk, m, c, d);\\
&\quad b' \leftarrow \textit{verify}(vk, m', c, d');\\
&\quad \textit{return}(b \wedge b')\}\ \textit{ELSE return}(\textit{False})
\end{aligned}$$

Definition 8 $\textit{binding-advantage}(\mathcal{A}) = \mathcal{P}[\textit{binding-game}(\mathcal{A}) = \textit{True}]$

Definition 9 $\textit{perfect-binding}(\mathcal{A}) = (\textit{binding-advantage}(\mathcal{A}) = 0)$

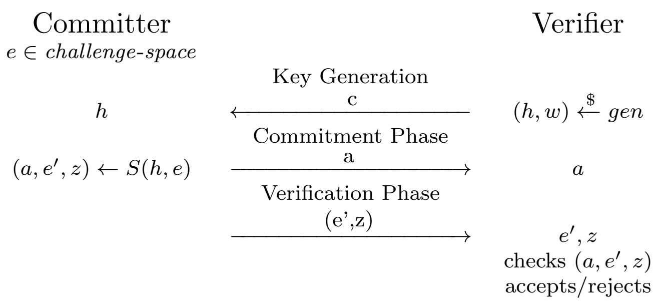

Fig. 8: A commitment scheme constructed from a Σ-protocol, m is the message being committed to.

8 Commitment Schemes from Σ-protocols

In this section we first describe the construction from [23] that uses a Σ-protocol to realise a commitment scheme that is perfectly hiding and computationally binding and then show how we formalise the construction at an abstract level. That is we fix a Σ-protocol and use its components to construct a commitment scheme and prove it secure. Realising the proof at a general level like this allows us to easily instantiate the result for the Σ-protocols we consider.

8.1 Constructing Commitment Schemes from Σ-protocols

Modern cryptography is based on hardness assumptions. These are relations that it is considered computationally infeasible to break. For example the discrete log assumption given in Equation 1.

Consider a hard relation R for a Σ-protocol such that gen generates h and w such that $R(h, w)$ is satisfied. Using a Σ-protocol for the relation R we can construct the commitment scheme given in Figure 8. In the key generation phase the verifier runs the generation algorithm, $(h, w) \leftarrow gen$ and sends h to the committer. To commit to a message e the committer runs the simulator on their key h and e; that is they run $(a, e', z) \leftarrow S(h, e)$ and send a to the verifier and keep e' and z as the opening values. 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 public input h.

Correctness comes from the HVZK property of the Σ-protocol, the simulator's output is the same as the output of a real execution of the protocol, meaning the check algorithm will accept the conversation. The commitment scheme is perfectly hiding because the commitment a is the first message of the Σ-protocol which is created independently of the challenge (the message being

```
locale $\Sigma$-$commit$ = $\Sigma$-$protocol$-$base$ $init^C$ $response^C$ $check^C$ $Rel^C$ $S^C_{raw}$ $\mathcal{A}^C_{ss}$
                                        $challenge$-$space^C$ $valid$-$pub^C$
    for $init^C$ $response^C$ $check^C$ $Rel^C$ $S^C_{raw}$ $\mathcal{A}^C_{ss}$ $challenge$-$space^C$ $valid$-$pub^C$+
    and $gen^C$
  assumes $\Sigma$-$protocol(h, w)$
    and $(h, w) \in set$-$spmf(gen^C) \Longrightarrow (h, w) \in Rel^C$
    and $lossless(gen^C)$
    and $lossless(init^C(h, w))$
    and $lossless(response^C(r, w, e))$
```

Fig. 9: The locale fixing the parameters of a Σ-protocol and the assumptions required to prove the commitment scheme construction.

committed to). The binding property follows from the special soundness property of the Σ-protocol; if the committer could output the commitment a and opening values (e, z) and (e', z') such that both (a, e, z) and (a, e', z') are both accepting conversations then by the special soundness property there exists an adversary that can output the witness w which contradicts the assumption on the relation being hard.

8.2 Formalising the construction

To formalise this construction we fix the components of a Σ-protocol in a locale and assume they form a Σ-protocol. The locale can be seen in Figure 9, where the superscript C denotes we are using the parameters to construct a commitment scheme. The only additional parameter we require in this construction beyond what the Σ-protocol provides is a generator,

$$gen^C \ :: \ (\text{`}pub\text{-}input \times \text{`}witness) \ spmf$$

that outputs (h, w) such that the relation is satisfied.

Using these fixed parameters we make the assumptions they form a Σ-protocol and that the generator outputs a tuple for which the relation holds. The assumptions on the lossessness of the parameters are needed, otherwise the protocol may terminate if they do not output anything; — meaning we cannot reason about the security properties.

To formalise the general notion of a hard relation we define a security game played by an adversary who is trying to break the relation: (h, w) is sampled from gen^C and h is given to the adversary who is asked to output w'. The adversary wins the game if $(h, w') \in Rel^C$.

```
$rel$-$game(\mathcal{A})$ = $TRY\ do\ \{$
  $(h, w) \leftarrow gen^C$;
  $w' \leftarrow \mathcal{A}(h)$;
  $return((h, w') \in Rel^C)\}\ ELSE\ return(False)$
```

Using this game we define the relation advantage — the probability an adversary has of winning the game.

Definition 10

$$\textit{rel-advantage}(\mathcal{A}) = \mathcal{P}[\textit{rel-game}(\mathcal{A}) = \textit{True}]$$

We show a reduction to this advantage in the proof of the binding property.

To formalise the protocol given in Figure 8 we define the three components $\textit{key-gen}^C$, $\textit{commit}^C$, $\textit{verify}^C$ that make up the commitment scheme and also what constitutes a valid message by defining $\textit{valid-msg}^C = (m \in \textit{challenge-space}^C)$. The keys are generated by sampling from $\textit{gen}^C$.

$$\begin{aligned}
&\textit{key-gen}^C = \textit{do}\ \{\\
&\quad (h, w) \leftarrow G^C;\\
&\quad \textit{return}(h, (h, w))\}
\end{aligned}$$

To commit to a message the committer runs the simulator and outputs the initial message from the simulator as the commitment and holds the response as the opening value.

$$\begin{aligned}
&\textit{commit}^C(h, e) = \textit{do}\ \{\\
&\quad (a, e, z) \leftarrow S^C(h, e);\\
&\quad \textit{return}(a, z)\}
\end{aligned}$$

Finally the verifier checks if the messages it has received from the committer correspond to an accepting conversation.

$$\textit{verify}^C((h, w), e, a, z) = \textit{check}^C(h, a, e, z)$$

We now prove that our construction of the commitment scheme meets the desired properties. The *commit-base* locale is imported under the name *Σ-commit* thus all definitions are prefixed with this.

sublocale *Σ-commit* : *commit-base key-gen*^{C} *commit*^{C} *verify*^{C} *valid-msg*^{C} .

The formal proofs of the security properties broadly follow the intuition given in Section 8.1. The proof sketches can be found in Appendix B. The correctness and hiding properties are given in Lemmas 8 and 9 below.

Lemma 8 *(in Σ-commit)* **shows** *Σ-commit.correct*

Lemma 9 *(in Σ-commit)* **shows** *Σ-commit.perfect-hiding*$(\mathcal{A})$

Finally we consider the binding property. Here we show a reduction to the relation advantage. To show this reduction we construct an adversary, $\textit{adversary}_{\textit{rel}}$, that interacts with the relation game using the Σ-protocols special soundness adversary and the adversary used in the binding game — $\textit{adversary}_{\textit{rel}}$ calls the binding adversary and constructs two conversations from it to pass them as inputs to the special soundness adversary and outputs the witness given.

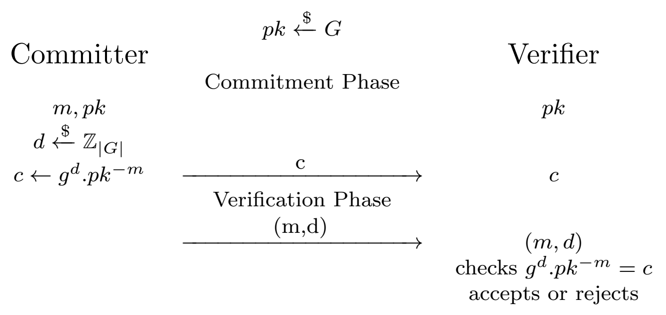

Fig. 10: The Pedersen commitment protocol, the committer commits to message m. No keys are known only to one party, we only have a publically known key pk.

$$\begin{aligned}&\mathit{adversary}_{rel}(\mathcal{A}, h) = \mathit{do}\ \{\\&\quad (c, e, z, e', z') \leftarrow \mathcal{A}(h);\\&\quad \mathcal{A}_{ss}^{C}(x, (c, e, z), (c, e', z'))\}\end{aligned}$$

Lemma 10 *(in Σ-commit)*
shows $\Sigma\text{-}\mathit{commit.bind\text{-}advantage}(\mathcal{A}) \leq \mathit{rel\text{-}advantage}(\mathit{adversary}_{rel}(\mathcal{A}))$

The next Section details how we use this general proof to realise the commitment schemes constructed from the Σ-protocols we consider — in particular we show how the security statements for the Pedersen commitment scheme come with very little proof effort.

9 The Pedersen Commitment Scheme

The Pedersen commitment scheme is a well known commitment scheme that allows for the commitment to a natural number. In [13] we formalised the Pedersen commitment scheme from scratch. In this work, our general proof of the construction of commitment schemes from Σ-protocols, from Section 8, gives the result in a matter of lines of proof.

We note the exact instantiation of the general result from Section 8 outputs a form of the Pedersen scheme that is slightly different from the traditional version presented. Specifically the commitment is taken as $c = g \cdot pk^{-m}$ rather than $c = g \cdot pk^{m}$ that is commonly presented in the literature, note the verification step is also modified in the analogous way. This is due to the simulator in the Schnorr protocol taking the inverse of the public input in constructing the initial message. The Pedersen protocol that arises from our formalisation is given in Figure 10.

Figure 11 shows the entire proof effort required to prove the Pedersen commitment scheme secure using our general proof from Section 8. First we import, under the name *pedersen*, the locale where the general proof is given and prove the import is valid. The correctness and perfect hiding properties come directly from the general proof, this is seen by the proof that only calls

```
sublocale pedersen:
  Σ_commit init response check R_DL S2 ss_adversary challenge_space G
  by unfold_locales
    (auto simp add: R_DL_def G_def Schnorr_Σ_inv.L_def sigma_protocol
     lossless_init lossless_response valid_pub_def)

lemma "pedersen.commit_base.correct"
  by(fact pedersen.commit_correct)

lemma "pedersen.commit_base.perfect_hiding_ind_cpa 𝒜"
  by(fact pedersen.perfect_hiding)

lemma rel_adv_eq_dis_log_adv:
  "pedersen.rel_advantage 𝒜 = dis_log.advantage 𝒜"
proof-
  have "pedersen.rel_game 𝒜 = dis_log.dis_log 𝒜"
    unfolding pedersen.rel_game_def R_DL_def dis_log.dis_log_def
    by(auto intro: try_spmf_cong bind_spmf_cong[OF refl]
       simp add: G_def cong_less_modulus_unique_nat group_eq_pow_eq_mod
       finite_carrier pow_generator_eq_iff_cong)
  thus ?thesis
    using pedersen.rel_advantage_def dis_log.advantage_def by simp
qed

lemma bind_advantage_bound_dis_log:
  "pedersen.commit_base.bind_advantage 𝒜 ≤ dis_log.advantage
(pedersen.adversary 𝒜)"
  using pedersen.bind_advantage rel_adv_eq_dis_log_adv by simp
```

Fig. 11: The proof (extracted from Isabelle) of the instantiation of the security statements for the Pedersen commitment scheme using the general proof of the construction of commitment schemes from Σ-protocols.

the on the lemmas *pedersen.correct-commit* and *pedersen.perfect-hiding* respectively. For the binding property in the general proof (Lemma 10) we show a reduction to the hard relation, in any instantiation we must relate this to the hardness assumption corresponding to the commitment scheme that has been constructed. In this case we show the relation advantage in the general construction is equivalent to the discrete log advantage. This is shown by the lemma *rel-adv-eq-dis-log-adv* in Figure 11. Using this we can show the binding advantage is bound by the discrete log advantage, thus completing the reduction for the binding property.

9.1 Asymptotic Security for the Pedersen and Schnorr protocols

So far, we have proved concrete security statements. Information-theoretic security notions like perfect hiding can be easily formalised in the concrete setting. Computational properties like computationally binding, however, can only be formalised in this setting by proving bounds in terms of hard problems. We now switch to the asymptotic security setting where we can formally express and prove computational security notions.

```
locale schnorr_asymp =
  fixes 𝒢 :: "nat ⇒ 'grp cyclic_group"
  assumes schnorr: "⋀η. schnorr_Σ_protocol (𝒢 η)"
begin

sublocale schnorr_Σ_protocol "𝒢 η" for η
  by(simp add: schnorr)

lemma Σ_protocol:
  shows "Schnorr_Σ.Σ_protocol n h w"
  by(simp add: sigma_protocol)

lemma asymp_correct: "pedersen.commit_base.correct n"
  using pedersen.commit_correct by simp

lemma asymp_perfect_hiding: "pedersen.commit_base.perfect_hiding n (𝒜 n)"
  using pedersen.perfect_hiding by blast

lemma asymp_computational_binding:
  assumes "negligible (λ n. dis_log.advantage n (pedersen.adversary n (𝒜 n)))"
  shows "negligible (λ n. pedersen.commit_base.bind_advantage n (𝒜 n))"
  using pedersen.bind_advantage assms pedersen.commit_base.bind_advantage_def
         negligible_le bind_advantage_bound_dis_log by auto

end
```

Fig. 12: Proving security in the asymptotic setting for the Schnorr Σ-protocol and the Pedersen commitment scheme.

To that end, we must introduce a security parameter n to the formalisation and make all definitions and statements depend on n. Then, we can easily derive the conventional asymptotic security statements from the concrete ones. We use Isabelle's locale instantiation mechanism as shown in Fig. 12 to achieve this with little effort. First we construct a locale that fixes the family of cyclic groups and then import the *schnorr-Σ-protocol* locale for all n. The statement that the Schnorr protocol is a Σ-protocol in the asymptotic setting comes trivially from the concrete setting (lemma *Σ-protocol*), as do the statements of correctness (*asymp-correct*) and perfect hiding (*asymp-perfect-hiding*) for the Pedersen commitment scheme.

It is left to show computational binding for the Pedersen commitment scheme. Here we show $\mathcal{A}$'s advantage against the binding game is negligible if *adversary*'s advantage against the discrete log game is negligible. This follows directly from the bound in the concrete case.

10 Further protocols and schemes

We have formalised more protocols beyond those discussed in the main part of this paper. The full outline of our formalisation is given in Figure 1. Here we briefly discuss the other protocols we formalise and point to the more detailed discussion of them in the appendix.

10.1 Compound Σ-protocols – the AND construction

In Section 6 we formalised how to construct a Σ-protocol for the OR of two statements. We have also formalised the corresponding construction for the AND of two statements. Like in the OR construction we let Σ_0 and Σ_1 be the underlying Σ-protocols. The relation Rel_{AND} is formally defined as:

$$Rel_{AND} = \{((x_0, x_1), (w_0, w_1)).\ (x_0, w_0) \in Rel_0 \wedge (x_1, w_1) \in Rel_1\}.$$

where Rel_0 and Rel_1 correspond to the relations of the two underlying Σ-protocols. Unlike in the OR construction we define this as a set rather than an inductive set.

The idea of the construction, Σ_{AND}, is simpler than the OR construction. The prover proves both statements in parallel for the same challenge sent by the verifier.

The formal proofs come more easily than in the OR construction as the underlying Σ-protocols are run in parallel, making it easier to use their respective security properties. The added complexity of the sum type needed in the OR construction is also not needed as the witness is a tuple (w_0, w_1) :: *'witness₀* $\times$ *'witness₁* rather than a single element that could either be of type *'witness₀* or *'witness₁*.

Our formalisation of the AND construction is given in Appendix C.

10.2 The Chaum-Pedersen and Okamoto Σ-protocols

The Chaum-Pedersen and Okamoto protocols are based on variations of the discrete log assumption. The Chaum-Pedersen protocol is based on the equality of discrete logarithms relation: $Rel_{CP} = \{((h_0, h_1), w).\ h_0 = g^w \wedge h_1 = g'^w\}$ whereas the Okamoto protocol is based on a relation whereby the public input is just h and the witness comprises as a tuple (w_0, w_1): $Rel_{Oka} = \{(h, (w_0, w_1)).\ (h = g^{w_0} \wedge h = g^{w_1})\}$ where g and g' are distinct generators of the cyclic group G.

Naturally both protocols are similar to the Schnorr protocol which is based on the discrete log assumption. Many similar arguments are used in the formal proof, especially in the rewriting of various terms. However, it was not always possible to reuse the exact auxiliary lemmas proven in the Schnorr protocol as the form of the group element constructions are subtly different in each case.

More details on our formalisation of the Chaum-Pedersen and Okamoto Σ-protocols are given in Appendices D and E respectively.

10.3 Rivest Commitment Scheme

The Rivest commitment scheme uses a trusted initialiser to distribute correlated randomness to both parties before the protocol is run. Its formalisation is of interest for two reasons.

Firstly, the trusted initialiser model is different from the standard form of a commitment scheme. So we must consider how to model it in our framework. We choose to model the distributed randomness sent to each party by the trusted initialiser as the keys each party holds in the execution of the protocol — specifically we define the key generation algorithm to output the randomness the trusted initialiser sends to the respective parties.

Secondly, the security results for the Rivest protocol are not obtained by the general result of commitment schemes from Σ-protocols proven in Section 8. This is because it is not based on any hardness assumption, and thus there is not an associated relation. Commitment schemes without a trusted initialiser cannot be both perfectly hiding and binding [24]. However as the Rivest protocol utilises a trusted initialiser, it can achieve both perfect hiding and binding and thus not rely on a hardness assumption.

Details of our formalisation of the Rivest commitment scheme can be found in Appendix F.

11 Related Work and Discussion

There are a number of tools that can be used for reduction based cryptographic proofs such as CertiCrypt [4], CryptHOL [6], EasyCrypt [3] and FCF [35]. These tools were all initially designed for game-based cryptographic proofs however some have been used for simulation-based proofs too; in [12] and [27] standalone protocols MPC protocols were considered whereas more recent work [32] and [16] considers composibility in the form of Constructive Cryptography and Universal Composibility respectively.

We highlight two reasons we believe the choice of using CryptHOL and Isabelle is justified. Firstly, as we have mentioned throughout this paper, CryptHOL provides a strong foundation to formalise cryptography in a modular way. This allows others to pick up and easily extend the work given here. For example if one wanted to extend the definitions of Σ-protocols to consider witness indistinguishablitiy then one can simply incorporate the definitions into the abstract theory and construct the instantiated proofs in the relevant places. Likewise, if one needed a Σ-protocol or commitment scheme, and its corresponding security properties, in a more complex protocol we have demonstrated how they can be assumed and general proofs constructed. Thus we feel CryptHOL goes a long way to providing the ability to formally reason about security proofs in the way they are often considered on paper, with a *cut and pasting* of properties of underlying primitives. While other frameworks for formalising cryptography have similar concepts — EasyCrypt has a theory cloning mechanism and CertiCrypt and FCF inherit the module system from

Coq — they are not used as extensively as in CryptHOL, for example they do not prove security in the asymptotic setting.

Secondly we highlight what is in our opinion an understated advantage of Isabelle — the archive of formal proofs (AFP). The AFP is a refereed collection of formalisations in Isabelle that is kept up to date for the current Isabelle release. In particular this ensures any formalisation accepted to the AFP can be used and added to with ease. Even if CryptHOL were not to be used for a number of years one could still download an up-to-date version compatible with the most recent Isabelle release at any point in the future. It is perhaps not quite as obvious how to do this with other frameworks for cryptography that do not have such support behind them. The AFP also means there is a vast infrastructure of mathematical libraries available to the user, this is especially relevant in our instantiations where the results rely heavily on the underlying number theory — much of which has been formalised already.

The drawback or barrier to entry to using CryptHOL is that one needs to understand Isabelle first. While this is not a trivial undertaking we suggest it is not considerably greater than learning the intricacies of any other formal cryptographic framework.

Commitment schemes have been studied before in EasyCrypt in [33] where the Pedersen commitment scheme was proven secure. One noticeable difference between the proof effort required is in the construction of the adversary used to prove computational binding — in particular in outputting the inverse of an element in a field. In EasyCrypt the inverse function is defined with the required property, that is: $x \neq 0 \Rightarrow x \cdot inv(x) = 1$ and consequently division is defined as $y \neq 0 \Rightarrow \frac{x}{y} = x \cdot inv(y)$. In Isabelle on the other hand we do not axiomatise the property of an inverse, but derive it from the Bezout function. This means our approach could be considered more foundational, and thus warrants the extra proof effort required.

Σ-protocols have been considered in [5] using CertiCrypt. The authors first proved secure a general construction of Σ^{ϕ}-protocols that prove knowledge of a preimage under a group homomorphism ϕ — the Schnorr and Okamoto Σ-protocols that we formalise are examples of this type. Secondly they considered the compound statements we formalise in Section 6. Their work however only considered the compound statements over bitstrings whereas our formalisation is over an arbitrary boolean algebra of which bitstrings of a given length are one instance.

Both [33] and [5] formalise some of the protocols we consider however they do so in different frameworks. For the ongoing development of the area we believe that it is important to have up-to-date and usable formalisations in the same framework; therefore we feel our work provides a strong basis for further formalisations in this area.

11.1 Differences in the definitions of Σ-protocols

There are different definitions of Σ-protocols presented in the literature [5,19, 20,22,29]. We now discuss their differences and the consequences of Cramer's additional HVZK requirement (Condition 2 in Definition 2). We also outline how Barthe et al. dealt with this issue in their formalisation of Σ-protocols [5].

Damgard's HVZK definition Damgard's definition [22] of HVZK does not require the inputs to the real view to satisfy the relation, namely it only requires that the output distributions of the simulator and real view are equal. We found two problems with this requirement. First, the real view is not well-defined if the public input is not in the relation: to construct the real view, we must run the prover and the prover runs only if it gets a witness as input, but there is no such witness when the public input is not in the relation. Accordingly, none of the proofs of HVZK for Σ-protocols we study work. For example, without the assumption that $h = g^w$ (from $(h, w) \in Rel^S$) in the Schnorr Σ-protocol, we cannot reason about the real view and the simulator being equal. In particular, we have no way of showing $a = g^z \cdot h^{-e}$ outputted by the simulator is equal to the initial message that is constructed in the real view. Second, Damgård assumes in the proofs in [22] that the relation holds for the input. We therefore conclude that Damgård probably intended to include restrict h to the restriction that $(h, w) \in Rel$ in his definition.

Hazay's and Lindell's HVZK definition In [29], Hazay and Lindell credit Damgard for providing the 'basis' of their presentation of Σ-protocols. Their definition requires the relation to be satisfied on the public input and witness that are inputs to the real view. This corresponds to condition 1 of Definition 2 in this work.

Damgård [22] and Hazay and Lindell [29] both carry out the OR construction for Σ-protocols with the relation Rel_{OR} as defined in Section 6.1, with a proof similar to ours. However, their proofs are flawed as the simulator for the HVZK property is unspecified for public inputs h that are not in the language. Accordingly, completeness need not hold.

Cramer's HVZK definition Cramer [20] additionally requires that the simulator outputs an accepting conversation when the public input is not in the language, which corresponds to Condition 2 in 2. This ensures that the completeness proof of the OR construction for Σ-protocols goes through. Lindell has confirmed that it was implicitly assumed in the proof [private communication, 2019]. We therefore conclude that the extended definition should be the standard one.

To our knowledge no real-world Σ-protocol violates the additional requirement — pathological examples can of course be constructed. In fact, it was straightforward to show the additional requirement for all the Σ-protocols we consider, yet this extended property is rarely required in the literature. However, it is crucial for the OR construction, which allows to efficiently prove compound statements in zero knowledge.

Barthe et al.'s formalisation and Ciampi et al.'s HVZK definition There is another way to rescue the OR construction without adding Cramer's requirement, namely changing the definition of Rel_{OR}. Barthe et al. [5] also noticed the completeness issue for the OR construction in their formalisation of Σ-protocols. They recovered the proof by defining Rel_{OR} as

$$\begin{aligned}\mathit{Rel}_{OR} = \{((x_0, x_1), w).\ &((x_0, w) \in \mathit{Rel}_0 \wedge x_1 \in \mathit{Domain}(\mathit{Rel}_1)) \\ &\vee ((x_1, w) \in \mathit{Rel}_1 \wedge x_0 \in \mathit{Domain}(\mathit{Rel}_0))\}, \quad (11)\end{aligned}$$

i.e., the both inputs x_0 and x_1 to be in the language. Ciampi et al. [19] use the same definition in their paper proofs.

In contrast, our definition (and Damgard's, Hazay's and Lindell's, and Cramer's) requires only one input x_0 or x_1 to be in the language; the other need only meet syntactic constraints as formalised by *valid-pub*. This small difference has a substantial impact on the expressive power of the OR construction. With (11), the languages for the constituent Σ-protocols must be *efficiently* decidable. Indeed, Ciampi et al. "implicitly assume that the verifier of a protocol for relation R executes the protocol only if the common input x belongs to L_R and rejects immediately common inputs not in L_R" [18]. For relations like the discrete logarithm, this is not a problem because every group element has a discrete logarithm; the hard part is computing it. However, there are Σ-protocols where the language itself is hard, e.g., Blum's protocol for a Hamiltonian cycle in a graph [9]. The OR construction with the relation (11) does not work for such Σ-protocols.

12 Conclusion

In this work we have formalised commitment schemes and Σ-protocols using the CryptHOL framework in Isabelle/HOL. The frameworks we provide are modular and thus can easily be used and extended by others. In principle the work we present could have been carried out in other formal frameworks for cryptography.

The merit of formalising cryptography is shown by the issue we uncover regarding the definition of Σ-protocols. While the cryptographer's intuition may usually suffice, it is important that the correct definitions are presented consistently in the literature.

Our work is limited as it cannot reason about polynomial runtime, a central concept in modern cryptography. CryptHOL cannot yet reason about runtime, however we hope to integrate it into our framework here when it becomes available. Without being able to express this efficiency notion the security definitions we provide must be considered without it, however due to the nature of our reductions this does not pose a significant problem. The main drawback is in the inability to formalise the hardness assumptions adequately — we cannot quantify over the set of all efficient adversaries, but only over all adversaries.

Consequently, incorporating the notion of run-time into our framework constitutes future work. Moreover a logical next step to increase the usability of our framework for others would be to define and reason about full Zero-Knowledge as this is an extension of the HVZK property of Σ-protocols. We believe this work is also likely to be of interest when formalising the malicious MPC security model as commitment schemes, Σ-protocols and Zero-Knowledge are commonly used to transfer protocols from semi-honest to malicious security.

Acknowledgements We would like to thank Yehuda Lindell for his help in arriving at the correct definition of Σ-protocols.

References

1. G Barthe, B Grégoire, and S Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *POPL*, pages 90–101. ACM, 2009.
2. 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.
3. Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago 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. Gilles Barthe, Benjamin Grégoire, and Santiago Zanella-Béguelin. Formal certification of code-based cryptographic proofs. In *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009*, pages 90–101. ACM, 2009.
5. Gilles Barthe, Daniel Hedin, Santiago Zanella Béguelin, Benjamin Grégoire, and Sylvain Heraud. A machine-checked formalization of sigma-protocols. In *CSF*, pages 246–260. IEEE Computer Society, 2010.
6. David A. Basin, Andreas Lochbihler, and S. Reza Sefidgar. CryptHOL: Game-based proofs in higher-order logic. *IACR Cryptology ePrint Archive*, page 753, 2017.
7. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, 2006.
8. Manuel Blum. Coin flipping by telephone. In *CRYPTO*, pages 11–15. U. C. Santa Barbara, Dept. of Elec. and Computer Eng., ECE Report No 82-04, 1981.
9. Manuel Blum. How to prove a theorem so no one else can claim it. In *International Congress of Mathematicians*, pages 1444–1451, 1986.
10. Carlo Blundo, Barbara Masucci, Douglas R. Stinson, and Ruizhong Wei. Constructions and bounds for unconditionally secure non-interactive commitment schemes. *Des. Codes Cryptogr.*, 26(1-3):97–110, 2002.
11. David Butler and David Aspinall. Multi-party computation. *Archive of Formal Proofs*, May 2019. `https://www.isa-afp.org/entries/Multi_Party_Computation.html`, Formal proof development.
12. David Butler, David Aspinall, and Adrià Gascón. How to simulate it in isabelle: Towards formal proof for secure multi-party computation. In *ITP*, volume 10499 of *Lecture Notes in Computer Science*, pages 114–130. Springer, 2017.
13. David Butler, David Aspinall, and Adrià Gascón. On the formalisation of Σ-protocols and commitment schemes. In *POST*, volume 11426 of *Lecture Notes in Computer Science*, pages 175–196. Springer, 2019.

14. David Butler and Andreas Lochbihler. Sigma protocols and commitment schemes. *Archive of Formal Proofs*, 2019. `https://www.isa-afp.org/entries/Sigma_Commit_Crypto.html`, Formal proof development.
15. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.
16. Ran Canetti, Alley Stoughton, and Mayank Varia. Easyuc: Using easycrypt to mechanize proofs of universally composable security. In *Proceedings of the 32nd IEEE Computer Security Foundations Symposium*, CSF 2019, Hoboken, NJ, USA, 2019. IEEE Computer Society.
17. David Chaum and Torben P. Pedersen. Wallet databases with observers. In *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 1992.
18. Michele Ciampi, Giuseppe Persiano, Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Improved OR-composition of Sigma-protocols. Cryptology ePrint Archive, Report 2015/810, 2015. `https://eprint.iacr.org/2015/810`.
19. Michele Ciampi, Giuseppe Persiano, Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Improved OR-composition of Sigma-protocols. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography*, pages 112–141. Springer, 2016.
20. R. Cramer. Modular design of secure, yet practical cryptographic protocols. *PhD thesisPhD Thesis, University of Amsterdam*, 1996.
21. Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 1994.
22. I. Damgard. On Σ-protocols. *Lecture Notes, University of Aarhus, Department for Computer Science.*, 2002.
23. Ivan Damgård. On the existence of bit commitment schemes and zero-knowledge proofs. In *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 17–27. Springer, 1989.
24. Ivan Damgård, Joe Kilian, and Louis Salvail. On the (im)possibility of basing oblivious transfer and bit commitment on weakened security assumptions. In *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 56–73. Springer, 1999.
25. Shimon Even. Protocol for signing contracts. In *CRYPTO*, pages 148–153. U. C. Santa Barbara, Dept. of Elec. and Computer Eng., ECE Report No 82-04, 1981.
26. Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.
27. Helene Haagh, Aleksandr Karbyshev, Sabine Oechsner, Bas Spitters, and Pierre-Yves Strub. Computer-aided proofs for multiparty computation with active security. In *CSF*, pages 119–131. IEEE Computer Society, 2018.
28. Shai Halevi. A plausible approach to computer-aided cryptographic proofs. *IACR Cryptology ePrint Archive*, 2005:181, 2005.
29. Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Information Security and Cryptography. Springer, 2010.
30. Andreas 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.
31. Andreas Lochbihler. CryptHOL. *Archive of Formal Proofs*, 2017.
32. Andreas Lochbihler, S. Reza Sefidgar, David A. Basin, and Ueli Maurer. Formalizing constructive cryptography using CryptHOL. In *Computer Security Foundations (CSF 2019)*, pages 152–166. IEEE, 2019.
33. Roberto Metere and Changyu Dong. Automated cryptographic analysis of the pedersen commitment scheme. In *MMM-ACNS*, volume 10446 of *Lecture Notes in Computer Science*, pages 275–287. Springer, 2017.
34. Tobias Nipkow and Gerwin Klein. *Concrete Semantics - With Isabelle/HOL*. Springer, 2014.
35. Adam Petcher and Greg Morrisett. The foundational cryptography framework. In *POST*, volume 9036 of *Lecture Notes in Computer Science*, pages 53–72. Springer, 2015.
36. Ronald Rivest. Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer. *Unpublished manuscript*, 1999.

37. Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.
38. Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive*, 2004:332, 2004.

A Proofs from OR Σ-protocol construction

Lemma 5 *(in Σ-OR-proof)* **shows** *Σ-OR.completeness*

Proof For ease we split the proof into cases depending on which relation holds. For the case where $Rel_1(x_1, w)$ holds the components corresponding to Rel_1 are generated using the Σ-protocol Σ_1, whereas the components corresponding to Rel_0 are simulated using S_0. For the correctly generated case (Rel_1) the check outputs true due to the completeness property of Σ_1. For the simulated case (Rel_0) we use the HVZK property (Condition 2) from Σ_0 to show the check outputs true. □

Lemma 6 *(in Σ-OR-proof)* **shows** *Σ-OR.HVZK*

Proof We simulate the real view by running the simulator (given in Equation 9) for both relations. The challenges we give to the simulators (e_0 and e_1) are related by $s = e_0 \oplus e_1$, where we sample e_1 uniformly (we could have sampled e_0) and s is the challenge in the OR construction. This asymmetry (we must sample one of e_0 or e_1) is dealt with using the lemma given in Equation 8. In the case where $Rel_0(x_0, w)$ holds the result comes directly by writing the components from Σ_0 in *Σ-OR.R* into the real view then using the HZVK property of Σ_0 to rewrite the real view as the simulator. In the case where $Rel_1(x_1, w)$ holds we follow the same process but use Equation 8 in the last step. □

Lemma 7 *(in Σ-OR-proof)* **shows** *Σ-OR.special-soundness*

Proof We must show $\mathcal{A}_{ss,OR}$ is lossless and always outputs a witness for Rel_{OR}. We have two conversations $((a_0, a_1), s, (e_0, z_0), (e_1, z_1))$ and $((a_0, a_1), s', (e'_0, z'_0), (e'_1, z'_1))$ on public inputs x_0 and x_1 respectively. We can assume the following hold (the assumptions in the statement of special soundness):

- $s \neq s'$
- $check_{OR}((x_0, x_1), (a_0, a_1), s, (e_0, z_0), (e_1, z_1))$
- $check_{OR}((x_0, x_1), (a_0, a_1), s', (e'_0, z'_0), (e'_1, z'_1))$
- $(x_0, x_1) \in valid\text{-}pub_{OR}$
- $s, s' \in challenge\text{-}space_{OR}$

From $s \neq s'$ we show that $e_0 \neq e'_0 \vee e_1 \neq e'_1$ and partition the proof on the case $e_0 \neq e'_0$. When this condition holds we know the conditions for the special soundness property for Σ_0 hold and thus $\mathcal{A}_{ss,0}$ is lossless and outputs a witness to Rel_0. The branch of the if statement that is invoked in $\mathcal{A}_{ss,OR}$ in this case calls $\mathcal{A}_{ss,0}$ and therefore outputs a witness to Rel_0. The proof for the second case, $e_1 \neq e'_1$, is analogous. □

B Proofs from Section 8

Lemma 8 *(in Σ-commit)* **shows** *Σ-commit.correct*

Proof We rewrite the simulator as the real view of the transcript using the HVZK property of Σ-protocols (Definition 3). After unfolding the real view into the components of the Σ-protocol we apply the definition of completeness (Definition 1) to show that check will always return true. □

Lemma 9 *(in Σ-commit)* **shows** *Σ-commit.perfect-hiding($\mathcal{A}$)*

```
$response_{AND}((r_0, r_1), (w_0, w_1), s) = do\ \{$
    $z_0 \leftarrow response_0(r_0, w_0, s);$
    $z_1 \leftarrow response_1(r_1, w_1, s);$
    $return(z_0, z_1)\}$

$check_{AND}((x_0, x_1), (a_0, a_1), s, (z_0, z_1)) =$
$(check_0(x_0, a_0, s, z_0) \wedge check_1(x_1, a_1, s, z_1))$
```

Fig. 13: The reponse and check functions for the AND construction.

```
$S_{AND}((x_0, x_1), e) = do\ \{$
    $(a_0, c_0, z_0) \leftarrow S_0(x_0, e);$
    $(a_1, c_1, z_1) \leftarrow S_1(x_1, e);$
    $return((a_0, a_1), e, (z_0, z_1))\}$

$\mathcal{A}_{ss,AND}((x_0, x_1), conv, conv') = do\ \{$
    $let\ ((a_0, a_1), e, (z_0, z_1)) = conv;$
    $let\ ((a_0', a_1'), e', (z_0', z_1')) = conv';$
    $w_0 \leftarrow \mathcal{A}_{ss,0}(x_0, (a_0, e, z_0), (a_0', e', z_0'));$
    $w_1 \leftarrow \mathcal{A}_{ss,1}(x_1, (a_1, e, z_1), (a_1', e', z_1'));$
    $return(w_0, w_1)\}$
```

Fig. 14: The special soundness adversary and simulator for the AND construction.

Proof We replace the simulator in the hiding game by the real view of the Σ-protocol. The commitment a comes from the probabilistic program $init^C$ and is therefore independent of the message that is committed as the only inputs to $init^C$ are h and w. Thus the adversary learns nothing of the committed message and so the chance of it winning the hiding game is equivalent to guessing the output of a coin flip — which implies perfect hiding. □

Lemma 10 *(in Σ-commit)*
shows $\Sigma\text{-}commit.bind\text{-}advantage(\mathcal{A}) \leq rel\text{-}advantage(adversary_{rel}(\mathcal{A}))$

Proof The binding game is equal to calling $rel\text{-}game(adversary_{rel})$ with the assertions from the binding game incorporated in the probabilistic program. When removing the assertions the probability mass of the probabilistic program can only increase, thus the bound in the above statement is valid. □

C AND construction for Σ-protocols

Section 6.1 showed how a Σ-protocol for the OR of two relations can be constructed. Here we show how this can be done for the AND of two relations.

The relation Rel_{AND} is defined as:

$$Rel_{AND} = \{((x_0, x_1), (w_0, w_1)).\ ((x_0, w_0) \in Rel_0 \wedge (x_1, w_1) \in Rel_1)\}.$$

The idea of the construction, Σ_{AND}, is more simple than the OR construction. The prover proves both statements in parallel for the same challenge sent by the verifier. The construction of the initial messages are shown below and the other components in Figures 13 and 14.

```
$init_{AND}((x_0, x_1), (w_0, w_1)) = do\ \{$
    $(r_0, a_0) \leftarrow init_0(x_0, w_0);$
    $(r_1, a_1) \leftarrow init_1(x_1, w_1);$
    $return((r_0, r_1), (a_0, a_1))\}$
```

The parallel running of both Σ_0 and Σ_1 can be seen easily here. Analogous to the case of the OR construction we import the Σ-protocol locale as $\Sigma\text{-}AND$. Due to the construction being more simple than the OR construction the proofs of correctness, HVZK and special soundness come more easily too. The proofs are able to directly use the corresponding properties of Σ_0 and Σ_1.

Lemma 11 *(in Σ-AND)* **shows** $\Sigma\text{-}AND.completeness$

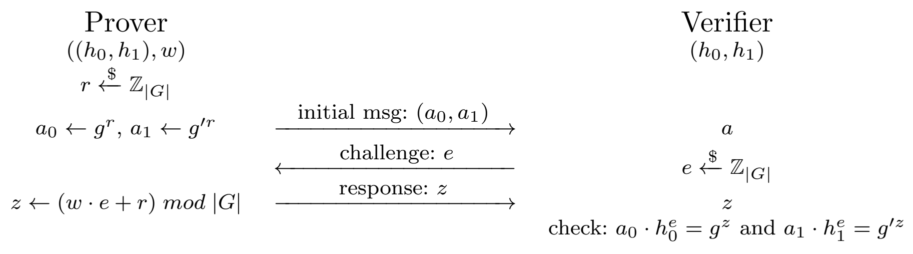
Prover
$((h_0, h_1), w)$
$r \xleftarrow{\$} \mathbb{Z}_{|G|}$
$a_0 \leftarrow g^r,\ a_1 \leftarrow g'^r$
initial msg: (a_0, a_1)
a
challenge: e
$e \xleftarrow{\$} \mathbb{Z}_{|G|}$
$z \leftarrow (w \cdot e + r)\ mod\ |G|$
response: z
z
Verifier
(h_0, h_1)
check: $a_0 \cdot h_0^e = g^z$ and $a_1 \cdot h_1^e = g'^z$

Fig. 15: The Chaum-Pedersen Σ-protocol.

Proof The executions of Σ_0 and Σ_1 are run in parallel, therefore the completeness properties of Σ_0 and Σ_1 can be applied straightforwardly for completeness to be realised. □

Lemma 12 *(in Σ-AND)* **shows** *Σ-AND.HVZK*

Proof The conversations for the AND construction are the conversations for Σ_0 and Σ_1 combined, thus both can be simulated by the HVZK property of Σ_0 and Σ_1, the simulator (given in Figure 14) does exactly this. □

Lemma 13 *(in Σ-AND)* **shows** *Σ-AND.special-soundness*

Proof The special soundness adversary, $\mathcal{A}_{ss,AND}$, runs the special soundness adversaries for both Σ_0 and Σ_1 to get the witnesses for each relation. The correct witnesses are outputted due to the adversaries for Σ_0 and Σ_1 outputting the correct witnesses for their respective protocols and $\mathcal{A}_{ss,AND}$ is lossless as the adversaries it uses are lossless, again due to the special soundness soundness property of Σ_0 and Σ_1. □

Combining the properties we can show the construction is a Σ-protocol.

Theorem 3 *(in Σ-AND)* **shows** *Σ-AND.Σ-protocol*

D Chaum-Pedersen Σ-protocol

In this section we detail our formalisation of the Chaum-Pedersen Σ-protocol [17]. The protocol is run over a cyclic group G of prime order where g and g' are generators of G. The relation considered here could be described as the equality of discrete logs relation.

$$Rel_{CP} = \{((h_0, h_1), w).\ h_0 = g^w \wedge h_1 = g'^w\} \tag{12}$$

The protocol is shown in Figure 15.

In the locale *chaum-ped-Σ-base* we fix the group G and a natural x that we use to construct $g' = g^x$.

```
locale chaum-ped-Σ-base =
  fixes G :: 'grp cyclic-group
    and x :: nat
  assumes prime(|G|)
begin
```

As usual we define the components of the Σ-protocol.

$init_{CP}((h_0, h_1), w) = do\ \{$
$\quad r \leftarrow samp\text{-}uniform(|G|);$
$\quad return(r, (g^r, g'^r))\}$

$check_{CP}((h_0, h_1), (a_0, a_1), e, z)$
$= (a_0 \otimes h_0^e = g^z \wedge a_1 \otimes h_1^e = g'^z)$

$response_{CP}(r, w, e) = (return(w \cdot e + r)\ mod\ |G|)$

```
$S_{CP}((h_0, h_1), e) = do \{$
    $z \leftarrow samp\text{-}uniform(|G|);$
    $let\ a = g^z \otimes (h_0^{-e});$
    $let\ a' = g'^z \otimes (h_1^{-e});$
    $return((a, a', e, z))\}$

$\mathcal{A}_{ss,CP}((h_0, h_1), c_1, c_2) = do \{$
    $let\ ((a, a'), e, z) = c_1;$
    $let\ ((b, b'), e', z') = c_2;$
    $return(if\ \ e > e'\ then\ (z - z') \cdot inv_G(e - e')$
                 $else\ (z' - z) \cdot inv_G(e' - e))\}$
```

Fig. 16: The simulator and the special soundness adversary for the Chaum-Pedersen Σ-protocol.

After importing the *Σ-protocol-base* locale as *CP-Σ* we construct a new locale where we import the cyclic group properties of G in which to prove the properties of the protocol.

locale *chaum-ped-Σ = chaum-ped-Σ-base + cyclic-group(G)*
begin

The unfolded simulator used to show HVZK and the special soundness adversary are given in Figure 16. Both the defining probabilistic programs, up to its inputs, are very similar to the adversary for the Schnorr Σ-protocol. This is to be expected as the relation and the protocol of the Chaum-Pedersen Σ-protocol are strongly related to the Schnorr Σ-protocol. The intuition behind the construction of the simulator is to uniformly sample the response to ensure it contains no information about the witness (by definition). The other components of the output can then be constructed around this uniform sample.

The proofs of the properties here are similar to the proofs of the Schnorr Σ-protocol (Lemmas 2, 3 and 4) the general difference being we do everything twice as we have two initial messages sent compared to one in the Schnorr protocol. The statements of the security properties are given below.

Lemma 14 *(in chaum-ped-Σ)* **shows** *CP-Σ.HVZK*

Lemma 15 *(in chaum-ped-Σ)* **shows** *CP-Σ.special-soundness*

Lemma 16 *(in chaum-ped-Σ)* **shows** *CP-Σ.completeness*

Together Lemmas 14, 15 and 16 imply our formalisation of the Chaum-Pedersen Σ-protocol is a Σ-protocol.

Theorem 4 *(in chaum-ped-Σ)* **shows** *CP-Σ.Σ-protocol*

E Okamoto Σ-protocol

In this section we detail our formalisation of the Okamoto Σ-protocol [17]. The protocol is run over a cyclic group G of prime order where g and g' are generators of G. The relation is as follows.

$$Rel_{Ok} = \{(h, (w_0, w_1)).\ h = g^{w_0} \otimes g'^{w_1}\} \tag{13}$$

The protocol is shown in Figure 17.

In the locale *okamoto-Σ-base* we fix the group G and a natural x that we use to construct $g' = g^x$, this is equivalent to the Chaum-Pedersen Σ-protocol.

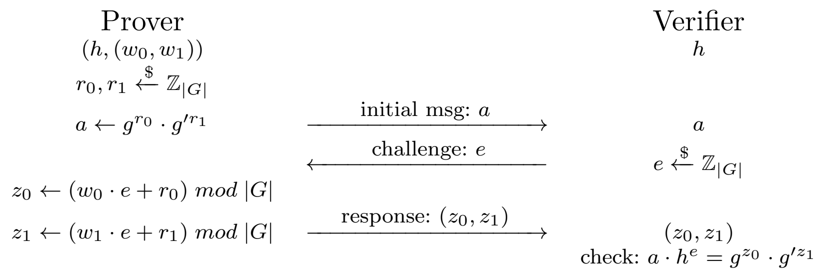
Prover
Verifier
$(h, (w_0, w_1))$
h
$r_0, r_1 \xleftarrow{\$} \mathbb{Z}_{|G|}$
$a \leftarrow g^{r_0} \cdot g'^{r_1}$
initial msg: a
a
challenge: e
$e \xleftarrow{\$} \mathbb{Z}_{|G|}$
$z_0 \leftarrow (w_0 \cdot e + r_0)\ mod\ |G|$
$z_1 \leftarrow (w_1 \cdot e + r_1)\ mod\ |G|$
response: (z_0, z_1)
(z_0, z_1)
check: $a \cdot h^e = g^{z_0} \cdot g'^{z_1}$

Fig. 17: The Okamoto Σ-protocol.

```
$S_{Ok}(h, e) = do$ {
  $z_0 \leftarrow samp\text{-}uniform(|G|);$
  $z_1 \leftarrow samp\text{-}uniform(|G|);$
  $let\ a = g^{z_0} \otimes g'^{z_1} \otimes (h^{-e});$
  $return(a, e, (z_0, z_1))$}

$\mathcal{A}_{ss,Ok}(h, c_1, c_2) = do$ {
  $let\ (a, e, (z_0, z_1)) = c_1;$
  $let\ (a', e', (z'_0, z'_1)) = c_2;$
  $return(if\ e > e'\ then\ (z_0 - z'_0) \cdot inv_G(e - e')$
          $else\ (z'_0 - z_0) \cdot inv_G(e' - e),$
              $if\ e > e'\ then\ (z_1 - z'_1) \cdot inv_G(e - e')$
                  $else\ (z'_1 - z_1) \cdot inv_G(e' - e))$}
```

Fig. 18: The simulator and the special soundness adversary for the Okamoto Σ-protocol.

```
locale okamoto-Σ-base =
  fixes G :: 'grp cyclic-group
    and x :: nat
  assumes prime(|G|)
begin
```

As usual we define the components of the Σ-protocol.

```
$init_{Ok}(h, w) = do$ {
  $r_0 \leftarrow samp\text{-}uniform(|G|);$
  $r_1 \leftarrow samp\text{-}uniform(|G|);$
  $return((r_0, r_1), (g^{r_0} \otimes g'^{r_1}))$}

$response_{Ok}((r_0, r_1), (w_0, w_1), e) =$
  $return(w_0 \cdot e + r_0)\ mod\ |G|, w_1 \cdot e + r_1)\ mod\ |G|$

$check_{Ok}(h, a, e, (z_0, z_1)) = (a \otimes h^e = g^{z_0} \otimes g'^{z_1})$
```

After importing the *Σ-protocol-base* locale as *O-Σ* we construct a new locale where we import the cyclic group properties of G in which to prove the properties of the protocol.

```
locale okamoto-Σ = okamoto-Σ-base + cyclic-group(G)
begin
```

The unfolded simulator used to show HVZK and the special soundness adversary are given in Figure 18.

The proofs of the properties here are similar to the proofs of the Schnorr Σ-protocol (Lemmas 4, 3 and 2) the general difference being we do everything twice as we have two initial messages sent compared to one in the Schnorr protocol — here we just give the statements of the properties.

Lemma 17 *(in okamoto-Σ)* **shows** *O-Σ.HVZK*

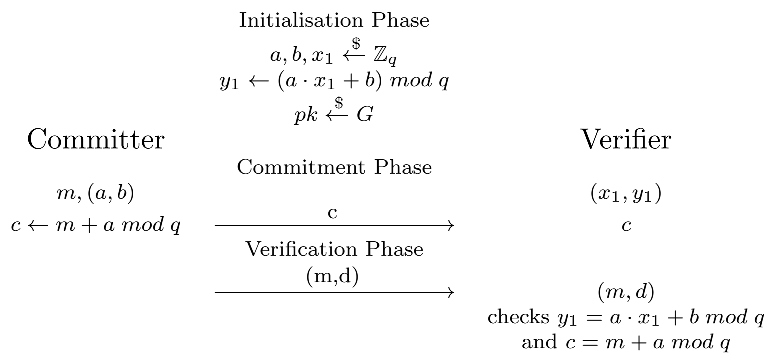

Fig. 19: The Affine Plane commitment scheme of [10] that slightly amends the Rivest commitment scheme [36].

Lemma 18 *(in okamoto-Σ)* **shows** *O-Σ.special-soundness*

Lemma 19 *(in okamoto-Σ)* **shows** *O-Σ.completeness*

Together Lemmas 17, 18 and 19 imply our formalisation of the Okamoto Σ-protocol is a Σ-protocol.

Theorem 5 *(in okamoto-Σ)* **shows** *O-Σ.Σ-protocol*

F Rivest Commitment Scheme

In this section we show how we formalise the Rivest commitment scheme [36]. The Rivest scheme is run using a field of prime order, $\mathbb{Z}_q$ and is built using a trusted initialiser. In this case the trusted initialiser provides co-related randomness to the parties in advance of the protocol, it does not participate in the running of the protocol thereafter. Protocols using a trusted initialiser are generally easier to implement as the initialisation can be performed in advance of the protocol and the co-related randomness reduces overheads in the protocol itself.

The protocol we formalise is shown in Figure 19. Note this is not quite the original scheme proposed by Rivest in [36]; as was noted by Blundo and Masucci in [10] the original scheme did not provide perfect hiding. The original committed message was constructed as $c = a \cdot m + b\ mod\ q$, the authors offered a slight amendment that does provide perfect hiding — it is this protocol we formalise in our work, and that is presented in Figure 19. The trusted initialiser randomly generates a, b and x_1 and constructs $y_1 = a \cdot x_1 + b\ mod\ q$. It sends (a, b) to the committer and (x_1, y_1) to the verifier. To commit to the message m the committer computes $c = m + a\ mod\ q$ and to reveal sends the pair (a, b) and the message m upon which the verifier checks $c = m + a\ mod\ q$ and $y_1 = a \cdot x_1 + b\ mod\ q$.

We formalise the protocol in the locale *rivest* where we fix the size of the field and assume it is of prime order. Note we do not use any field construction previously formalised in Isabelle, preferring to work modulo q throughout the formalisation.

```
locale rivest =
  fixes q :: nat
  assumes prime(q)
begin
```

The components of the commitment scheme are given in Figure 20. Our formalisation allows for the trusted initialiser as we treat the co-related randomness given to each party

```
$key\text{-}gen_R = do$ {
  $a \leftarrow samp\text{-}uniform(q)$;
  $b \leftarrow samp\text{-}uniform(q)$;
  $b \leftarrow samp\text{-}uniform(q)$;
  $let\ y_1 = (a \cdot x_1 + b)\ mod\ q$
  $return((a, b), (x_1, y_1))$}
```

```
$commit_R((a, b), m) = return(m + a\ mod\ q, (a, b))$

$verify_R((x_1, y_1), m, c, (a, b)) =$
    $(c = m + a\ mod\ q \wedge y_1 = a \cdot x_1 + b\ mod\ q)$

$valid\text{-}msg_R(m) = m \in \{1, \ldots, q - 1\}$
```

Fig. 20: The formalised components of the Rivest commitment scheme.

as the keys, the work done by the trusted initialiser in the protocol is done in our key generation algorithm. As usual we import the commitment scheme locale, here under the name *rivest-commit*.

We first consider the hiding property.

Lemma 20 *(in rivest)* **shows** *rivest-commit.perfect-hiding*$(\mathcal{A})$

Proof The commitment $c = m + a\ mod\ q$ reveals no information about m as it is masked by the randomness of a, which the verifier does not have access to. Therefore an application of the one time pad lemma for addition in a field (Equation 14), which we prove, means the committed message given to the adversary is independent of the message.

$$map(\lambda.\ (c + a)\ mod\ q, samp\text{-}uniform(q)) = samp\text{-}uniform(q) \tag{14}$$

We then show the adversary's guess can be no better than a than flipping a coin to determine its output, meaning its chance of winning the hiding game is $\frac{1}{2}$. □

The binding property is proven by bounding the binding advantage by $\frac{1}{q}$.

Lemma 21 *(in rivest)* **shows** *rivest-commit.bind-advantage*$(\mathcal{A}) \leq \frac{1}{q}$

Proof The conditions required on the output of the binding adversary (in the binding game) are such that we can compute x_1 (let us call the function computing x_1, f), which is uniformly sampled in the game (as part of the key generation algorithm), from the output of $\mathcal{A}$. Intuitively this means we can correctly guess the output of a uniform sampling from a set of q elements, the probability of which is $\frac{1}{q}$. More formally we have $f(a, a; , b, b') = x_1$ where x_1 is a uniform sample. As f is independent of x_1 we show the probability of the game returning true is less than or equal to f guessing the value of x_1, that is the probability is less than $\frac{1}{q}$. □

Correctness comes easily after unfolding the relevant definitions.

Lemma 22 *(in rivest)* **shows** *rivest-commit.correctness*

Together Lemmas 20, 21 and 22 show the desired properties of the commitment scheme presented in Figure 19.

G Roadmap to source theory files

Our formal proofs are available online at [14]. Below we give a guide to the reader to help navigate the formal theories.

- **Commitment_Schemes.thy** formalises commitment schemes (Section 7).
- **Sigma_protocols.thy** formalises Σ-protocols as well as the construction that forms a commitment scheme from a Σ-protocol (Section 4).

- **Pedersen.thy**, **Rivest.thy** formalise the Pedersen and Rivest commitments schemes respectively (Section 9 and Appendix F)[6]
- **Schnorr_Sigma_Commit.thy**, **Chaum_Pedersen_Sigma_Commit.thy** and **Okamoto_Sigma_Commit.thy** formalise the Schnorr, Chaum-Pedersen and Okamoto Σ-protocols as well as the instantiated proofs that they can be used to construct a commitment scheme.
- **Sigma_OR.thy**, **Sigma_AND.thy** formalise the compound Σ-protocol statements (Section 6.1 and Appendix C).
- **Xor.thy** formalises the concept of a boolean algebra, used in the OR and AND Σ-protocol construction.
- **Uniform_Sampling.thy** formalises numerous one time pad constructions used in our proofs.
- **Cyclic_Group_Ext.thy** extends the formalisation of cyclic groups from CryptHOL, providing results we require in this work.
- **Discrete_Log.thy** formalises the discrete log assumption as well as a variant (and a reduction from this to the original) that we require.
- **Number_Theory_Aux.thy** formalises various results from number theory we require, in particular we prove who we compute the inverse using the Bezout function — Lemma 1.

[6] The security statements for the Pedersen commitment scheme are obtained from the instantiation of the general Σ-protocol to commitment scheme construction using the Schnorr Σ-protocol. However the commitment scheme constructed there is subtly different to the traditional Pedersen commitment scheme as noted in Section 9. Therefore we keep our original formalisation (from scratch) of the Pedersen Scheme as well as the instantiated proof which appears in **Schnorr_Sigma_Commit.thy**.