The Wrestlers Protocol
A simple, practical, secure, deniable protocol for key-exchange

Mark Wooding
mdw@distorted.org.uk

2 November 2006

Abstract We describe and prove (in the random-oracle model) the security of a simple but efficient zero-knowledge identification scheme, whose security is based on the computational Diffie-Hellman problem. Unlike other recent proposals for efficient identification protocols, we don't need any additional assumptions, such as the Knowledge of Exponent assumption.

From this beginning, we build a simple key-exchange protocol, and prove that it achieves 'SK-security' – and hence security in Canetti's Universal Composability framework.

Finally, we show how to turn the simple key-exchange protocol into a slightly more complex one which provides a number of valuable 'real-life' properties, without damaging its security.

Contents

1 Introduction

This paper proposes protocols for *identification* and *authenticated key-exchange*.

An identification protocol allows one party, say Bob, to be sure that he's really talking to another party, say Alice. It assumes that Bob has some way of recognising Alice; for instance, he might know her public key. Our protocol requires only two messages – a challenge and a response – and has a number of useful properties. It is very similar to, though designed independently of, a recent protocol by Stinson and Wu; we discuss their protocol in section 3.4.

Identification protocols are typically less useful than they sound. As Shoup [Sho99] points out, it provides a 'secure ping', by which Bob can know that Alice is 'there', but provides no guarantee that any other communication was sent to or reached her. However, there are situations where this an authentic channel between two entities – e.g., a device and a smartcard – where a simple identification protocol can still be useful.

An authenticated key-exchange protocol lets Alice and Bob agree on a shared secret, known to them alone, even if there is an enemy who can read and intercept all of their messages, and substitute messages of her own. Once they have agreed on their shared secret, of course, they can use standard symmetric cryptography techniques to ensure the privacy and authenticity of their messages.

1.1 Desirable properties of our protocols

Our identification protocol has a number of desirable properties.

- It is *simple* to understand and implement. In particular, it requires only two messages.
- It is fairly *efficient*, requiring two scalar multiplications by each of the prover and verifier.
- It is provably *secure* (in the random oracle model), assuming the intractability of the computational Diffie-Hellman problem.

Our key-exchange protocol also has a number of desirable properties.

- It is fairly *simple* to understand and implement, though there are a few subtleties. In particular, it is *symmetrical*. We have implemented a virtual private network system based on this protocol.
- It is *efficient*, requiring four scalar multiplications by each participant. The communication can be reduced to three messages by breaking the protocol's symmetry.
- It is provably *secure* (again, in the random oracle model), assuming the intractability of the computational Diffie-Hellman problem, and the security of a symmetric encryption scheme.
- It provides *perfect forward secrecy*. That is, even if a user's long-term secrets are compromised, past sessions remain secure.
- It is *deniable*. It is possible to construct simulated transcripts of protocol executions between any number of parties without knowing any of their private keys. The simulated transcripts are (almost) indistinguishable from real protocol transcripts. Hence, a transcript does not provide useful evidence that a given party was really involved in a given protocol execution.

1.2 Asymptotic and concrete security results

Most security definitions for identification (particularly zero-knowledge) and key-exchange in the literature are *asymptotic*. That is, they consider a family of related protocols, indexed by a *security parameter* k; they that any *polynomially-bounded* adversary has only *negligible* advantage. A polynomially-bounded adversary is one whose running time is a bounded by some polynomial $t(k)$. The security definition requires that, for any such polynomially-bounded adversary, and any polynomial $p(k)$, the adversary's advantage is less than $p(k)$ for all sufficiently large values of k.

Such asymptotic notions are theoretically interesting, and have obvious connections to complexity theory. Unfortunately, such an asymptotic result tells us nothing at all about the security of a particular instance of a protocol, or what parameter sizes one needs to choose for a given level of security against a particular kind of adversary. Koblitz and Menezes [KM06] (among other useful observations) give examples of protocols, proven to meet asymptotic notions of security, whose security proofs guarantee nothing at all for the kinds of parameters typically used in practice.

Since, above all, we're interested in analysing a practical and implemented protocol, we follow here the 'practice-oriented provable security' approach largely inspired by Bellare and Rogaway, and exemplified by [BKR94, BGR95, BR95a, BR96, BCK96, BDJR97]; see also [Bel99]. Rather than attempting to say, formally, whether or not a protocol is 'secure', we associate with each protocol an 'insecurity function' which gives an upper bound on the advantage of any adversary attacking the protocol within given resource bounds.

1.3 Formal models for key-exchange

Many proposed key-exchange protocols have turned out to have subtle security flaws. The idea of using formal methods to analyse key-exchange protocols begins with the logic of Burrows, Abadi and Needham [BAN89]. Their approach requires a 'formalising' step, in which one expresses in the logic the contents of the message flows, and the *beliefs* of the participants.

Bellare and Rogaway [BR94] describe a model for studying the computational security of authentication and key-exchange protocols in a concurrent setting, i.e., where multiple parties are running several instances of a protocol simultaneously. They define a notion of security in this setting, and show that several simple protocols achieve this notion. Their original paper dealt with pairs of parties using symmetric cryptography; they extended their definitions in [BR95b] to study three-party protocols involving a trusted key-distribution centre.

Blake-Wilson, Johnson and Menezes [BWJM97] applied the model of [BR94] to key-exchange protocols using asymmetric cryptography, and Blake-Wilson and Menezes [BWM98] applied it to protocols based on the Diffie-Hellman protocol.

The security notion of [BR94] is based on a *game*, in which an adversary nominates a *challenge session*, and is given either the key agreed by the participants of the challenge session, or a random value independently sampled from an appropriate distribution. The adversary's advantage – and hence the insecurity of the protocol – is measured by its success probability in guessing whether the value it was given is really the challenge key. This challenge-session notion was also used by the subsequent papers described above.

Bellare, Canetti and Krawczyk [BCK98] described a pair of models which they called the

AM (for 'authenticated links model') and UM ('unauthenticated links model'). They propose a modular approach to the design of key-exchange protocols, whereby one first designs a protocol and proves its security in the AM, and then applies a authenticating 'compiler' to the protocol which they prove yields a protocol secure in the realistic UM. Their security notion is new. They define an 'ideal model', in which an adversary is limited to assigning sessions fresh, random and unknown keys, or matching up one session with another, so that both have the same key. They define a protocol to be secure if, for any adversary A in the AM or UM, there is an ideal adversary I, such that the outputs of A and I are computationally indistinguishable.

In [Sho99], Shoup presents a new model for key-exchange, also based on the idea of simulation. He analyses the previous models, particularly [BR94] and [BCK98], and highlights some of their inadequacies.

Canetti and Krawczyk [CK01] describe a new notion of security in the basic model of [BCK98], based on the challenge-session notion of [BR94]. The security notion, called 'SK-security', seems weaker in various ways than those of earlier works such as [BR94] or [Sho99]. However, the authors show that their notion suffices for constructing 'secure channel' protocols, which they also define.

In [Can01], Canetti describes the 'universal composition' framework. Here, security notions are simulation-based: one defines security notions by presenting an 'ideal functionality'. A protocol securely implements a particular functionality if, for any adversary interacting with parties who use the protocol, there is an adversary which interacts with parties using the ideal functionality such that no 'environment' can distinguish the two. The environment is allowed to interact freely with the adversary throughout, differentiating this approach from that of [BCK98] and [Sho99], where the distinguisher was given only transcripts of the adversary's interaction with the parties. With security defined in this way, it's possible to prove a 'universal composition theorem': one can construct a protocol, based upon various ideal functionalities, and then 'plug in' secure implementations of the ideal functionalities and appeal to the theorem to prove the security of the entire protocol. The UC framework gives rise to very strong notions of security, due to the interactive nature of the 'environment' distinguisher.

Canetti and Krawczyk [CK02] show that the SK-security notion of [CK01] is *equivalent* to a 'relaxed' notion of key-exchange security in the UC framework, and suffices for the construction of UC secure channels.

The result of [CK02] gives us confidence that SK-security is the 'right' notion of security for key-exchange protocols. Accordingly, SK-security is the standard against which we analyse our key-exchange protocol.

1.4 Outline of the paper

The remaining sections of this paper are as follows.

- Section 2 provides the essential groundwork for the rest of the paper. It introduces important notation, and describes security notions and intractability assumptions.
- Section 3 describes our zero-knowledge identification protocol and proves its security.
- Section 4 describes the simple version of our key-exchange protocol, and proves its security and deniability. It also describes some minor modifications which bring practical benefits without damaging security.

- Finally, section 5 presents our conclusions.

2 Preliminaries

2.1 Miscellaneous notation

We write $\mathcal{F}[D \to R]$ for the set of all functions with domain D and range R. We write $\mathbb{N}_{<n} = \{\, i \in \mathbb{Z} \mid 0 \leqslant i < n \,\} = \{0, 1, \ldots, n-1\}$ for the set of nonnegative integers less than n.

2.2 Groups

Let $(G, +)$ be a cyclic group[1]of prime order q, and generated by an element P. We shall write the identity of G as 0_G, or simply as 0 when no ambiguity is likely to arise. Thus, we have $\langle P \rangle = G$ and $qP = 0$. Any $X \in G$ can be written as $X = xP$ for some $x \in \mathbb{N}_{<q}$.

2.3 Bit strings and encodings

Let $\Sigma = \{0, 1\}$ be the set of binary digits. Then Σ^n is the set of n-bit strings, and Σ^* the set of all (finite) bit strings. If $x \in \Sigma^n$ is a bit string, we write its length as $|x| = n$. For a bit string $x \in \Sigma^n$, and for $0 \leqslant i < n$, we write $x[i]$ as the ith bit of x. The empty string is denoted λ.

Let x and y be two bit strings. If $|x| = |y| = n$, we write $x \oplus y$ to mean the bitwise exclusive-or of x and y: if $z = x \oplus y$ then $|z| = n$, and $z[i] = (x[i] + y[i]) \bmod 2$ for $0 \leqslant i < n$. We write $x \parallel y$ to mean the concatenation of x and y: if $z = x \parallel y$ then $|z| = |x| + |y|$ and $z[i] = x[i]$ if $0 \leqslant i < |x|$ and $z[i] = y[i - |x|]$ if $|x| < i \leqslant |x| + |y|$.

Finally, we let $\perp$ be a value distinct from any bit string.

We shall want to encode group elements $X \in G$ and indices $x \in I = \mathbb{N}_{<|G|}$ as bit strings. To this end, we shall assume the existence of integers $\ell_G, \ell_I > 0$ and functions

$$e_S\colon S \to \Sigma^{\ell_S} \quad \text{and} \quad d_S\colon \Sigma^{\ell_S} \to S \cup \{\perp\} \qquad \text{for } S \in \{G, I\}.$$

with the following properties.

- The functions are *unique* and *unambiguous*, i.e., for any $t \in \Sigma^{\ell_S}$, we have

$$d_S(t) = \begin{cases} s & \text{if there is some } s \in S \text{ such that } t = e_S(s) \text{, or} \\ \perp & \text{if no such } s \text{ exists.} \end{cases}$$

- The functions should be *efficient* to compute. Indeed, we shall be assuming that the time taken for encoding and decoding is essentially trivial.

Note that, as we have defined them, all encodings of group elements are the same length, and similarly for encodings of indices. This is necessary for the security of our protocols.

We shall frequently abuse notation by omitting the encoding and decoding functions where it is obvious that they are required.

[1] We find that additive group notation is easier to read. In particular, in multiplicative groups, one ends up with many interesting things tucked away in little superscripts.

2.4 Games, adversaries, and oracles

Many of the security definitions and results given here make use of *games*, played with an *adversary*. An adversary is a probabilistic algorithm. In some games, the adversary is additionally equipped with *oracles*, which perform computations with values chosen by the adversary and secrets chosen by the game but not revealed to the adversary. We impose limits on the adversary's resource usage: in particular, the total time it takes, and the number of queries it makes to its various oracles. Throughout, we include the size of the adversary's program as part of its 'time', in order to model adversaries which contain large precomputed tables.

The games provide models of someone trying to attack a construction or protocol. For security, we will either define a notion of 'winning' the game, and require that all adversaries have only a very small probability of winning, or we consider two different games and require that no adversary can distinguish between the two except with very small probability.

Our proofs make frequent use of sequences of games; see [Sho04, BR04]. The presentation owes much to Shoup [Sho04]. We begin with a game $\mathbf{G}_0$ based directly on a relevant security definition, and construct a sequence of games $\mathbf{G}_1$, $\mathbf{G}_2$, ..., each slightly different from the last. We define all of the games in a sequence over the same underlying probability space – the random coins tossed by the algorithms involved – though different games may have slightly differently-defined events and random variables. Our goal in doing this is to bound the probability of the adversary winning the initial game $\mathbf{G}_0$ by simultaneously (a) relating the probability of this event to that of corresponding events in subsequent games, and (b) simplifying the game until the probability of the corresponding event can be computed directly.

The following simple lemma from [Sho01] will be frequently useful.

2.4.1 Lemma (Difference Lemma) *Let S, T, F be events. Suppose $\Pr[S|\bar{F}] = \Pr[T|\bar{F}]$. Then $|\Pr[S] - \Pr[T]| \leqslant \Pr[F]$.*

Proof A simple calculation:

$$\begin{aligned} |\Pr[S] - \Pr[T]| &= |(\Pr[S|F]\Pr[F] + \Pr[S|\bar{F}]\Pr[\bar{F}]) - (\Pr[T|F]\Pr[F] + \Pr[T|\bar{F}]\Pr[\bar{F}])| \\ &= \Pr[F] \cdot |\Pr[S|F] - \Pr[T|F]| \\ &\leqslant \Pr[F] \end{aligned}$$

and we're done! □

2.5 The random oracle model

In particular, most of our results will make use of the *random oracle* model [BR93], in which all the participants, including the adversary, have access to a number of 'random oracles'. A random oracle with domain D and range R is an oracle which computes a function chosen uniformly at random from the set of all such functions. (In the original paper [BR93], random oracles are considered having domain Σ^* and range Σ^ω; we use finite random oracles here, because they're easier to work with.)

Given a protocol proven secure in the random oracle model, we can instantiate each random oracle by a supposedly-secure hash function and be fairly confident that either our protocol will be similarly secure, or one of the hash functions we chose has some unfortunate property.

Proofs in the random oracle must be interpreted carefully. For example, Canetti, Goldreich and Halevi [CGH04] show that there are schemes which can be proven secure in the random oracle model but provably have no secure instantiation in the standard model.

The random oracle model is useful for constructing reductions and simulators for two main reasons.

1. One can use the transcript of an adversary's queries to random oracles in order to extract knowledge from it.
2. One can 'program' a random oracle so as to avoid being bound by prior 'commitments', or to guide an adversary towards solving a selected instance of some problem.

Our proofs only make use of the first feature. This becomes particularly important when we consider issues of zero-knowledge and deniability in a concurrent setting, because we want to be able to claim that we retain these features when the random oracle is instantiated using a cryptographic hash function, and hash functions definitely aren't 'programmable' in this way! The former property seems rather more defensible – one would indeed hope that the only sensible way of working out (anything about) the hash of a particular string is to actually compute the hash function, and the random oracle model is, we hope, just giving us a 'hook' into this process.

(Our protocols can be modified to make use of bilinear pairings so as to provide identity-based identification and key-exchange, using the techniques of [BF03]. Proving the security of the modifications we discuss would involve 'programming' random oracles, but this doesn't affect the zero-knowledge or deniability of the resulting protocols.)

2.6 Notation for algorithms

We shall have occasion to describe algorithms by means of a pseudocode. Our choice of pseudocode is unlikely to be particularly controversial. We let $x \leftarrow y$ denote the action of setting x to the value y; similarly, $x \xleftarrow{\$} Y$ denotes the action of sampling x from the set Y uniformly at random.

The expression $a \leftarrow A^{O(\cdot,x)}(y)$ means 'assign to a the value output by algorithm A on input y, and with oracle access to the algorithm which, given input z, computes $O(z, x)$'.

We make use of conditional (**if-else**) and looping (**for-do** and **while-do**) constructions; in order to reduce the amount of space taken up, the bodies of such constructions are shown by indentation only.

We don't declare the types of our variables explicitly, assuming that these will be obvious by inspection; also, we don't describe our variables' scopes explicitly, leaving the reader to determine these from context.

Finally, the notation $\Pr[$*algorithm* : *condition*$]$ denotes the probability that *condition* is true after running the given *algorithm*.

2.7 Diffie-Hellman problems

The security of our protocols is related to the hardness of the computational, decisional, and gap Diffie-Hellman problems in the group G. We define these problems and what it means for them to be 'hard' here.

The *computational* Diffie-Hellman problem (CDH) is as follows: given two group elements $X = xP$ and $Y = yP$, find $Z = xyP$.

2.7.1 Definition (The computational Diffie-Hellman problem) Let $(G, +)$ be a cyclic group generated by P. For any adversary A, we say that A's *success probability* at solving the computational Diffie-Hellman problem in G is

$$\mathbf{Succ}_G^{\mathrm{cdh}}(A) = \Pr[x \xleftarrow{\$} I; y \xleftarrow{\$} \mathbb{N}_{<|G|} : A(xP, yP) = xyP]$$

where the probability is taken over the random choices of x and y and any random decisions made by A. We say that the *CDH insecurity function* of G is

$$\mathbf{InSec}^{\mathrm{cdh}}(G; t) = \max_A \mathbf{Succ}_G^{\mathrm{cdh}}(A)$$

where the maximum is taken over adversaries which complete in time t. □

Certainly, if one can compute discrete logarithms in the group G (i.e., given xP, find x), then one can solve the computational Diffie-Hellman problem. The converse is not clear, though. Shoup [Sho97] gives us some confidence in the difficulty of the problem by showing that a *generic* adversary – i.e., one which makes no use of the specific structure of a group – has success probability no greater than $q^2/|G|$.

This isn't quite sufficient for our purposes. Our proofs will be able to come up with (possibly) a large number of guesses for the correct answer, and at most one of them will be correct. Unfortunately, working out which one is right seems, in general, to be difficult. This is the *decision* Diffie-Hellman problem (DDH), which [Sho97] shows, in the generic group model, is about as hard as CDH. (See [Bon98] for a survey of the decision Diffie-Hellman problem.)

Our reference problem will be a 'multiple-guess computational Diffie-Hellman problem' (MCDH), which is captured by a game as follows. An adversary is given a pair of group elements (xP, yP), and an oracle $V(\cdot)$ which accepts group elements as input. The adversary wins the game if it queries $V(xyP)$.

2.7.2 Definition (The multiple-guess computational Diffie-Hellman problem) Let $(G, +)$ be a cyclic group generated by P. For some adversary A, we say that A's *success probability* at solving the multiple-guess computational Diffie-Hellman problem in G is

$$\mathbf{Succ}_G^{\mathrm{mcdh}}(A) = \Pr[\mathbf{Game}_G^{\mathrm{mcdh}}(A) = 1]$$

where $\mathbf{Game}_G^{\mathrm{mcdh}}(A)$ is shown in figure 2.1. We say that the *MCDH insecurity function of G* is

$$\mathbf{InSec}^{\mathrm{mcdh}}(G; t, q_V) = \max_A \mathbf{Succ}_G^{\mathrm{mcdh}}(A)$$

where the maximum is taken over adversaries which complete in time t and make at most q_V-oracle queries. □

Note that our MCDH problem is not quite the 'gap Diffie-Hellman problem' (GDH). The gap problem measures the intractibility of solving CDH even with the assistance of an oracle for solving (restricted) decision Diffie-Hellman problems in the group. Specifically, the adversary is given $(X, Y) = (xP, yP)$ and tries to find $Z = xyP$, as for CDH, but now it has access to an oracle $D(R, S)$ which answers 1 if $S = xR$ and 0 otherwise.

Clearly MCDH is at least as hard as GDH, since our simple verification oracle $V(Z)$ can be simulated with the gap problem's DDH oracle, as $D(Y, Z)$. However, we can (loosely) relate the difficulty of MCDH to the difficulty of CDH.

$$
\begin{array}{l|l}
\begin{array}{l}
\mathbf{Game}^{\mathrm{mcdh}}_G(A): \\
\quad w \leftarrow 0; \\
\quad x \xleftarrow{\$} \mathbb{N}_{<|G|};\ y \xleftarrow{\$} \mathbb{N}_{<|G|}; \\
\quad A^{V(\cdot)}(xP, yP); \\
\quad \textbf{return } w;
\end{array}
&
\begin{array}{l}
\text{Function } V(Z): \\
\quad \textbf{if } Z = xyP \textbf{ then} \\
\quad\quad w \leftarrow 1; \\
\quad\quad \textbf{return } 1; \\
\quad \textbf{return } 0;
\end{array}
\end{array}
$$

Figure 2.1: The multiple-guess computational Diffie-Hellman problem: $\mathbf{Game}^{\mathrm{mcdh}}_G(A)$

2.7.3 Proposition (Comparison of MCDH and CDH security) *For any cyclic group* $(G, +)$*,*

$$\mathbf{InSec}^{\mathrm{mcdh}}(G; t, q_V) \leqslant q_V \, \mathbf{InSec}^{\mathrm{cdh}}(G; t + O(q_V)).$$

Proof Let A be an adversary attacking the multiple-guess computational Diffie-Hellman problem in G, and suppose that it runs in time t and issues q_V queries to its verification oracle.

We use a sequence of games. Game $\mathbf{G}_0$ is the original MCDH attack game. In each game $\mathbf{G}_i$, we let the event S_i be the probability that the adversary wins the game.

Game $\mathbf{G}_1$ is the same as $\mathbf{G}_0$, except that we change the behaviour of the verification oracle. Specifically, we make the oracle always return 0. We claim that this doesn't affect the adversary's probability of winning, i.e., $\Pr[S_1] = \Pr[S_0]$. To see this, note that if none of the adversary's $V(\cdot)$ queries was correct, then there is no change in the game; conversely, if any query was correct, then the adversary will have won regardless of its subsequent behaviour (which may differ arbitrarily between the two games).

We are now ready to construct from A an adversary B attacking the standard computational Diffie-Hellman problem.

$$
\begin{array}{l|l}
\begin{array}{l}
\text{Adversary } B(X, Y): \\
\quad n \leftarrow 0; \\
\quad \textbf{for } i \in \mathbb{N}_{<q_V} \textbf{ do } Q_i \leftarrow 0; \\
\quad A^{V(\cdot)}; \\
\quad r \xleftarrow{\$} \mathbb{N}_{<n}; \\
\quad \textbf{return } Q_r;
\end{array}
&
\begin{array}{l}
\text{Function } D(Z'): \\
\quad Q_n \leftarrow Z'; \\
\quad n \leftarrow n + 1; \\
\quad \textbf{return } 0;
\end{array}
\end{array}
$$

Observe that B provides A with an accurate simulation of game $\mathbf{G}_1$. Moreover, at the end of the algorithm, we have $0 < n \leqslant q_V$, the output of A is stored in Q_{n-1} and the values Q_0, Q_1, …, Q_{n-1} are the values of A's oracle queries. Hence, with probability $Pr[S_1]$, at least of one of the Q_i is the correct answer to the CDH problem. Let $\varepsilon = \Pr[S_1] = \Pr[S_0]$; we claim that B's probability of success is at least ε/q_V. The proposition follows directly from this claim and that, because A was chosen arbitrarily, we can maximize and count resources.

We now prove the above claim. For $0 \leqslant i < q_V$, let W_i be the event that $Q_i = xyP$, i.e., that Q_i is the correct response. A simple union bound shows that

$$\sum_{0 \leqslant i < j} \Pr[W_i \mid n = j] \geqslant \varepsilon.$$

We now perform a calculation:

$$\begin{aligned}
\mathbf{Succ}^{\mathrm{cdh}}_G(B) &= \sum_{0 \leqslant i < q_V} \Pr[W_i \wedge r = i] \\
&= \sum_{0 < j \leqslant q_V} \Pr[n = j] \biggl(\sum_{0 \leqslant i < j} \Pr[W_i \wedge r = i \mid n = j] \biggr) \\
&= \sum_{0 < j \leqslant q_V} \Pr[n = j] \biggl(\frac{1}{j} \sum_{0 \leqslant i < j} \Pr[W_i \mid n = j] \biggr) \\
&\geqslant \sum_{0 < j \leqslant q_V} \Pr[n = j] \frac{\varepsilon}{j} \\
&\geqslant \frac{\varepsilon}{q_V} \sum_{0 < j \leqslant q_V} \Pr[n = j] \\
&= \frac{\varepsilon}{q_V}.
\end{aligned}$$

which completes the proof. □

2.8 Example groups and encodings

For nonnegative integers $0 \leqslant n < 2^\ell$, there is a natural binary encoding $N_\ell \colon \mathbb{N}_{<2^\ell} \to \Sigma^\ell$ which we can define recursively as follows.

$$N_0(0) = \lambda \qquad N_\ell(n) = \begin{cases} N_{\ell-1}(n) \parallel 0 & \text{if } 0 \leqslant n < 2^{\ell-1} \\ N_{\ell-1}(n - 2^{\ell-1}) \parallel 1 & \text{if } 2^{\ell-1} \leqslant n < 2^\ell. \end{cases}$$

Given an encoding $a = N_\ell(n)$ we can recover n as

$$n = \sum_{0 \leqslant i < \ell} a[i] 2^i.$$

Hence, given some limit $L \leqslant 2^\ell$, we can encode elements of $\mathbb{N}_{<L}$ using the functions (e, d):

$$e(L, \ell, n) = N_\ell(n) \qquad d(L, \ell, a) = \begin{cases} N_\ell(a) & \text{if } N_\ell(a) < L \\ \perp & \text{otherwise} \end{cases}$$

The reader can verify that the functions $e(L, \ell, \cdot)$ and $d(L, \ell, \cdot)$ satisfy the requirements of section 2.3.

Given some $q < 2^{\ell_I}$ and $I = \mathbb{N}_{<q}$, then, we can define an encoding (e_I, d_I) by $e_I(n) = e(q, \ell_I, n)$ and $d_I(a) = d(q, \ell_I, a)$.

Let p and q be primes, with $q | (p - 1)$. Then there is an order-q subgroup of $(\mathbb{Z}/p\mathbb{Z})^*$. In practice, an order-$q$ element can be found easily by taking elements $h \in (\mathbb{Z}/p\mathbb{Z})^*$ at random and computing $g = h^{(p-1)/2}$ until $g \neq 1$; then $G = \langle g \rangle$ is a group of q elements. Assuming that p and q are sufficiently large, the Diffie-Hellman problems seem to be difficult in G. Some texts recommend additional restrictions on p, in particular that $(p-1)/2q$ be either prime or the product of large primes. Primes of this form protect against small-subgroup attacks; but our protocols are naturally immune to these attacks, so such precautions are unnecessary here. Elements of G can be encoded readily, since each element $n + p\mathbb{Z}$ of $\mathbb{Z}/p\mathbb{Z}$ has an obvious 'representative' integer n such that $0 \leqslant n < p$, and given $2^{\ell_G} > p$, we can encode n as $e(p, \ell_G, n)$, as above.

Alternatively, let $\mathbb{F} = \mathbb{F}_{p^f}$ be a finite field, and E be an elliptic curve defined over $\mathbb{F}$ such that the group $E(\mathbb{F})$ of $\mathbb{F}$-rational points of E has a prime-order cyclic subgroup G. Elements of G can be represented as pairs of elements of $\mathbb{F}$. If $f = 1$, i.e., $\mathbb{F} = \mathbb{Z}/p\mathbb{Z}$ then field elements can be encoded as above. If $p = 2$, we can represent the field as $\mathbb{F}_2/(p(x))$ for some irreducible polynomial $p(x) \in \mathbb{F}_2[x]$ of degree f. An element $r \in \mathbb{F}$ can then be represented by a polynomial $r(x)$ with degree less than f, and coefficients $c_i \in \{0, 1\}$, i.e.,

$$r(x) = \sum_{0 \leqslant i < f} c_i x^i$$

and hence we can uniquely encode r as an f-bit string a such that $a[i] = c_i$.

2.9 Symmetric encryption

Our key-exchange protocol requires a symmetric encryption scheme. Our definition is fairly standard, except that, rather than specifying a key-generation algorithm, we assume that key generation simply involves selecting a string of a given length uniformly at random.

2.9.1 Definition (Symmetric encryption schemes) A *symmetric encryption scheme* $\mathcal{E} = (\kappa, E, D)$ consists of:

- an integer $\kappa \geqslant 0$,
- a randomized *encryption algorithm* E which, on input $K \in \Sigma^\kappa$ and $p \in \Sigma^*$ outputs some $c \in \Sigma^*$, written $c \leftarrow E_K(p)$;
- a *decryption algorithm* D which, on input $K \in \Sigma^\kappa$ and $c \in \Sigma^*$ outputs some $p' \in \Sigma^* \cup \{\perp\}$, written $p' \leftarrow D_K(c)$.

Furthermore, a symmetric encryption scheme must be *sound*: that is, if $c \leftarrow E_K(p)$ for some $K \in \Sigma^\kappa$ and $p \in \Sigma^*$, and $p' \leftarrow D_K(c)$ then $p = p'$. □

Our security notion for symmetric encryption is the standard notion of left-or-right indistinguishability of ciphertexts under chosen-ciphertext attack.

2.9.2 Definition (Indistinguishability under chosen-ciphertext attack (IND-CCA)) Let $\mathcal{E} = (\kappa, E, D)$ be a symmetric encryption scheme, and A be an adversary. Let $lr_b(x_0, x_1) = x_b$ for $b \in \{0, 1\}$. Let

$$P_b = \Pr[K \xleftarrow{\$} \Sigma^\kappa; b \leftarrow A^{E_K(lr_b(\cdot,\cdot)), D_K(\cdot)}() : b = 1]$$

An adversary is *valid* if

- for any query to its encryption oracle $E_K(lr_b(x_0, x_1))$ we have $|x_0| = |x_1|$, and
- no query to the decryption oracle $D_K(\cdot)$ is equal to any reply from an encryption query.

If A is valid, then we define its *advantage* in attacking the security of $\mathcal{E}$ as follows

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{E}} = P_1 - P_0.$$

Further, we define the *IND-CCA insecurity function of $\mathcal{E}$* to be

$$\mathbf{InSec}^{\text{ind-cca}}(\mathcal{E}; t, q_E, q_D) = \max_A \mathbf{Adv}^{\text{ind-cca}}_{\mathcal{E}}(A)$$

where the maximum is taken over all valid adversaries A which run in time t, and issue at most q_E encryption and q_D decryption queries. □

2.10 Simulations

In section 3, we shall prove that our identification protocol is zero-knowledge; in section 4.5, we show that our key-exchange protocol is deniable. In both of these proofs, we shall need to demonstrate *simulatability*.

General framework Consider a game in which an adversary A interacts with some 'world' W, which we shall represent as a probabilistic algorithm. The world may in fact represent a number of honest parties communicating in a concurrent fashion, but we can consider them as a single algorithm for our present purposes.

Initially the world and the adversary are both given the same *common input* c; in addition, the world is given a *private input* w. Both c and w are computed by an *initialization function* I, which is considered to be part of the definition of the game. Finally, the adversary decides somehow that it has finished interacting, and outputs a value a. All this we notate as

$$(w, c) \leftarrow I(); a \leftarrow A^{W(w,c)}(c).$$

This game is *simulatable* if there is an algorithm S – the *simulator* – which can compute the same things as A, but all by itself without interacting with the world. That is, we run the simulator on the common input c, allowing it to interact in some way with the adversary A, and finally giving us an output s.

$$(w, c) \leftarrow I(); s \leftarrow S^A(c).$$

We shall say that the simulator is *effective* if it's difficult to tell whether a given string was output by the adversary after interacting with the world, or by the simulator running by itself. That is, for any algorithm D – a *distinguisher* – running in some bounded amount of time, its advantage

$$\Pr[(w, c) \leftarrow I(); a \leftarrow A^{W(w,c)}(c); b \leftarrow D(c, a) : b = 1] - \\ \Pr[(w, c) \leftarrow I(); s \leftarrow S^A(c); b \leftarrow D(c, s) : b = 1]$$

is small. (Note that we gave the distinguisher the common input as well as the output of the adversary or the simulator.)

It's usual to study *transcripts* of interactions in these kinds of settings. We are considering arbitrary adversarial outputs here, so this certainly includes adversaries which output a transcript of their interactions. Indeed, for any adversary A, we could construct an adversary A_T which performs the same computation, and outputs the same result, but also includes a complete transcript of A's interaction with the world. Therefore we're just providing additional generality.

Random oracles We shall be considering interactions in which all the parties have access to several random oracles. We could simply say that the random oracles are part of the world W. In the setting described above, only the adversary actually interacts with the world (and therefore would be able to query random oracles). The simulator would be forced to 'make up' its own random oracle, and the distinguisher would have to study the distributions of the random-oracle queries and their responses to make up its mind about which it was given.

However, this would be a poor model for the real world, since once we instantiate the random oracle with a hash function, we know that everyone would in actually be able to compute the hash function for themselves. Thus a distinguisher in the real world would be able to tell the difference immediately between a real interaction and the simulated transcript, since the 'random oracle' queries recorded in the latter would be wrong!

Therefore we decide not to include the random oracles as part of the world, but instead allow all the participants – adversary, simulator and distinguisher – access to them. If we denote by $\mathcal{H} = (H_0, H_1, \ldots, H_{n-1})$ the collection of random oracles under consideration, the expression for the distinguisher's advantage becomes

$$\Pr[(w, c) \leftarrow I(); a \leftarrow A^{W(w,c),\mathcal{H}}(c); b \leftarrow D^{\mathcal{H}}(c, a) : b = 1] - \\ \Pr[(w, c) \leftarrow I(); s \leftarrow S^{A,\mathcal{H}}(c); b \leftarrow D^{\mathcal{H}}(c, s) : b = 1].$$

Auxiliary inputs If an adversary's output can be effectively simulated, then we can confidently state that the adversary 'learnt' very little of substance from its interaction, and certainly very little it can *prove* to anyone else. However, as we have described the setting so far, we fix an adversary before we choose inputs to the world, so our model says little about what an adversary which has already acquired some knowledge might learn beyond that. For example, an adversary might overhear some other conversation between honest parties and be able to use this to its advantage.

To this end, we give the adversary an *auxiliary input* u, computed by an algorithm U. We give U both c and w, in order to allow the adversary to gain some (possibly partial) knowledge of the secrets of the other parties. We also allow U access to the random oracles $\mathcal{H}$, because clearly in the 'real world' it would be ridiculous to forbid such an algorithm from computing a publicly-known hash function.

The simulator and distinguisher are also given the auxiliary input. The simulator is meant to represent the adversary's ability to compute things on its own, without interacting with the world, and since the adversary is given the auxiliary input, the simulator must be too. The distinguisher must be the auxiliary input because otherwise the simulator could just 'make up' plausible-looking inputs.

Resource limits We shall not allow our algorithms to perform completely arbitrary computations and interactions. Instead, we impose limits on the amount of time they are allowed to take, the number of random-oracle queries they make, and so on. Specifically, we are interested in

- the time t_A taken by the adversary and t_D taken by the distinguisher,
- the number of oracle queries $\mathcal{Q}_A = (q_{A,0}, q_{A,1}, \ldots, q_{A,n-1})$ made by the adversary, and $\mathcal{Q}_D$ made by the distinguisher,
- a number of resource bounds $\mathcal{R}$ on the adversary's interaction with the world (e.g., number of messages of various kinds sent and received), and
- a number of bounds $\mathcal{U}$ on the contents of the adversary's auxiliary input u.

Sometimes we shall not be interested in proving simulatability of adversaries with auxiliary inputs. We write $\mathcal{U} = 0$ to indicate that auxiliary output is not allowed.

World syntax It will be worth our while being more precise about what a 'world' actually is, syntactically. We define a world to be a single, randomized algorithm taking inputs $(\iota, \sigma, \tau, \mu) \in (\Sigma^*)^4$; the algorithm's output is a pair $(\sigma', \rho) \in (\Sigma^*)^2$. We show how the adversary's interaction is mapped on to this world algorithm in figure 2.2.

- The 'input' ι is the result of the initialization function I. That is, it is the pair (w, c) of the world's private input and the common input.
- The 'state' σ is empty on the world's first invocation; on each subsequent call, the value of the world's output σ' is passed back. In this way, the world can maintain state.
- The 'type τ is a token giving the type of invocation this is.
- The 'message' μ is any other information passed in; its form will typically depend on the type τ of the invocation.
- The 'new state' σ' is the value of σ to pass to the next invocation of the world.
- The 'reply ρ is the actual output of the invocation.

There are two special invocation types. The adversary is *forbidden* from making special invocations.

- The special invocation type init is used to allow the world to prepare an initial state. The world is invoked as
$$W^{\mathcal{H}}(\iota, \lambda, \mathsf{init}, \lambda)$$
and should output an initial state σ'. The world's reply ρ is ignored. (Recall that λ represents the empty string.)
- The special invocation type random is used to inform the world that the adversary has issued a random oracle query. The world is invoked as
$$W^{\mathcal{H}}(\iota, \sigma, \mathsf{random}, (i, x, h))$$
to indicate that the adversary has queried its random oracle $H_i(\cdot)$ on the input x, giving output h. The world may output an updated state σ'; its reply ρ is ignored.

The latter special query is a technical device used to allow the 'fake-world' simulators we define below to be aware of the adversary's random oracle queries without being able to 'program' the random oracle. Including it here does little harm, and simplifies the overall exposition.

Definitions We are now ready to begin making definitions.

2.10.1 Definition (Simulation security) Consider the game described above, with the initialization function I, and the world W: let A be an adversary, and let U be an auxiliary-input function; let S be a simulator, and let D be a distinguisher. We define D's *advantage against S's simulation of A's interaction with W with auxiliary inputs provided by U* to be

$$\mathbf{Adv}^{\mathrm{sim}}_{W,I,S}(A, U, D) = \Pr[\mathbf{Game}^{\mathrm{real}}_{W,I,S}(A, U, D) = 1] - \Pr[\mathbf{Game}^{\mathrm{sim}}_{W,I,S}(A, U, D) = 1]$$

where the games are as shown in figure 2.3. Furthermore, we define the *simulator's insecurity function* to be

$$\mathbf{InSec}^{\mathrm{sim}}(W, I, S; t_D, t_A, \mathcal{Q}_D, \mathcal{Q}_A, \mathcal{R}, \mathcal{U}) = \max_{D,A,U} \mathbf{Adv}^{\mathrm{sim}}_{W,I,S}(A, U, D)$$

Interaction $A^{W(w,c),\mathcal{H}}(c,u)$:
$(\sigma,\rho) \leftarrow W((w,c),\lambda,\mathsf{init},\lambda)$;
$a \leftarrow A^{\mathit{world}(\cdot,\cdot),\mathit{random}(\cdot,\cdot)}(c,u)$;
return a;

Function $\mathit{world}(\tau,\mu)$:
if $\tau \in \{\mathsf{init},\mathsf{random}\}$ **then return** $\perp$;
$(\sigma,\rho) \leftarrow W((w,c),\sigma,\tau,\mu)$;
return ρ;

Function $\mathit{random}(i,x)$:
$h \leftarrow H_i(x)$;
$(\sigma,\rho) \leftarrow W((w,c),\sigma,\mathsf{random},(i,x,h))$;
return h;

Figure 2.2: Interacting with a world: Interaction $A^{W,\mathcal{H}}$

$\mathbf{Game}^{\mathrm{real}}_{W,I,S}(A,U,D)$:
$(w,c) \leftarrow I()$;
$u \leftarrow U^{\mathcal{H}}(w,c)$;
$a \leftarrow A^{W(w,c),\mathcal{H}}(c,u)$;
$b \leftarrow D^{\mathcal{H}}(c,u,a)$;
return b;

$\mathbf{Game}^{\mathrm{sim}}_{W,I,S}(A,U,D)$:
$(w,c) \leftarrow I()$;
$u \leftarrow U^{\mathcal{H}}(w,c)$;
$s \leftarrow S^{A,\mathcal{H}}(c,u)$;
$b \leftarrow D^{\mathcal{H}}(c,u,s)$;
return b;

Figure 2.3: Games for simulation: $\mathbf{Game}^{\mathrm{real}}_{W,I}$ and $\mathbf{Game}^{\mathrm{sim}}_{W,I}$

where the maximum is taken over all distinguishers D running in time t_D and making at most $\mathcal{Q}_D$ random-oracle queries, and all adversaries A running in time t_A, making at most $\mathcal{Q}_A$ random-oracle queries, not exceeding the other stated resource bounds $\mathcal{R}$ on its interaction with W, and auxiliary-input functions producing output not exceeding the stated bounds $\mathcal{U}$. □

2.10.2 Remark The usual definitions of zero-knowledge, for example, require the simulator to work for all choices of inputs (common, private and auxiliary), rather than for random choices. Our definition therefore looks weaker. Our proof of zero-knowledge actually carries through to the traditional stronger-looking definition. Critically, however, the standard universal quantification over inputs fails to capture deniability in the random oracle model, since the inputs can't therefore depend on the random oracle. Our formulation therefore actually gives *stronger* deniability that the usual one. □

Fake-world simulators The simulators we shall be considering in the present paper are of a specific type which we call 'fake-world simulators'. They work by running the adversary in a fake 'cardboard cut-out' world, and attempting to extract enough information from the adversary's previous interactions and random oracle queries to maintain a convincing illusion.

That is, the behaviour of a fake-world simulator S is simply to allow the adversary to interact with a 'fake world' W', which was not given the world private input. That is, there is some world W' such that

$$S^{A,\mathcal{H}}(c,u) \equiv A^{W'(u,c),\mathcal{H}}(c,u)$$

Fake-world simulators are convenient because they allow us to remove from consideration the distinguisher D as the following definition shows.

2.10.3 Definition (Fake-world simulation security) Let I, W and U be as in definition 2.10.1. Let A be an adversary which outputs a single bit. Let S be a fake-world simulator. We define A's *advantage against S's fake-world simulation of W with auxiliary inputs provided by U* to be

$$\begin{aligned}\mathbf{Adv}^{\mathrm{fw}}_{W,I,S}(A,U) = \Pr[(w,c) \leftarrow I(); u \leftarrow U^{\mathcal{H}}(w,c); b \leftarrow A^{W(w,c),\mathcal{H}}(c,u) : b = 1] - \\ \Pr[(w,c) \leftarrow I(); u \leftarrow U^{\mathcal{H}}(w,c); b \leftarrow S^{A,\mathcal{H}}(c,u) : b = 1]\end{aligned}$$

Furthermore, we define the *simulator's insecurity function* to be

$$\mathbf{InSec}^{\mathrm{fw}}(W,I,S;t_D,t,\mathcal{Q},\mathcal{R},\mathcal{U}) = \max_{A,U} \mathbf{Adv}^{\mathrm{fw}}_{W,I,S}(A,U)$$

where the maximum is taken over all adversaries A running in time t, making at most $\mathcal{Q}$ random-oracle queries, not exceeding the other stated resource bounds $\mathcal{R}$ on its interaction with W, and auxiliary-input functions producing output not exceeding the stated bounds $\mathcal{U}$. □

It remains for us to demonstrate that this is a valid way of analysing simulators; the following simple proposition shows that this is indeed the case.

2.10.4 Proposition (Fake-world simulation) *Let I be an initialization function and let W be a world. Then, for any fake-world simulator S,*

$$\mathbf{InSec}^{\mathrm{sim}}(W,I,S;t_D,t_A,\mathcal{Q}_D,\mathcal{Q}_A,\mathcal{R},\mathcal{U}) \leqslant \mathbf{InSec}^{\mathrm{fw}}(W,I,S;t_A+t_D,\mathcal{Q}_D+\mathcal{Q}_A,\mathcal{R},\mathcal{U})$$

(where addition of query bounds $\mathcal{Q}$ is done elementwise).

Proof Let W and I as in the proposition statement be given; also let a distinguisher D running in time t_D and making $\mathcal{Q}_D$ random-oracle queries, an adversary A running in time t_A and making $\mathcal{Q}_A$ random-oracle queries and interacting with its world within the stated bounds $\mathcal{R}$, an auxiliary-input function U satisfying the constraints $\mathcal{U}$ on its output, and a fake-world simulator S all be given.

We construct an adversary B outputting a single bit as follows

Adversary $B^{W,\mathcal{H}}(c,u)$:
  $a \leftarrow A^{W,\mathcal{H}}(c,u)$;
  $b \leftarrow D^{\mathcal{H}}(c,u,a)$;
  return b;

A glance at definitions 2.10.1 and 2.10.3 and the resources used by B shows that

$$\mathbf{Adv}^{\mathrm{sim}}_{W,I,S}(A,U) = \mathbf{Adv}^{\mathrm{fw}}_{W,I,S}(B,U) \leqslant \mathbf{InSec}^{\mathrm{fw}}(W,I,S;t_D+t_A,\mathcal{Q}_D+\mathcal{Q}_A,\mathcal{R},\mathcal{U})$$

as required. □

3 A zero-knowledge identification scheme

3.1 Description

Here we present a simple zero-knowledge identification scheme. Fix some group G. Suppose Alice chooses a private key $x \in_{\$} \mathbb{N}_{<|G|}$, and publishes the corresponding public key $X = xP$.

Setup	Group $G = \langle P \rangle$; $\lvert G \rvert = q$ is prime. $H_I(\cdot, \cdot)$ is a secure hash.
Private key	$x \in_\$ \mathbb{N}_{<q}$.
Public key	$X = xP$.
Challenge	(R, c) where $r \in_\$ \mathbb{N}_{<q}$, $R = rP$, $c = r \oplus H_I(R, rX)$.
Response	$xR = rX$ if $R = (c \oplus H_I(R, xR))P$; otherwise $\perp$.

Figure 3.1: Summary of the Wrestlers Identification Protocol, W-ident

```
Function setup():
  x ←$ N_{<q};
  X ← xP;
  return (x, X);

Function challenge^{H_I(·,·)}(R, c, X):
  r ←$ N_{<q};
  R ← rP; Y ← rX;
  h ← H_I(R, Y); c ← r ⊕ h;
  return (Y, R, c);

Function verify(Y, Y'):
  if Y' = Y then return 1;
  return 0;

Function response^{H_I(·,·)}(R, c, x):
  Y' ← xR;
  h ← H_I(R', Y'); r' ← c ⊕ h;
  if R ≠ r'P then return ⊥;
  return Y';
```

Figure 3.2: Functions implementing W-ident in the random oracle model

Let $H_I \colon G^2 \to \Sigma^{\ell_I}$ be a secure hash function. Here's a simple protocol which lets her prove her identity to Bob.

1. Bob selects a random $r \in_\$ \mathbb{N}_{<\lvert G \rvert}$, and computes $R = rP$, $Y = rX$, and $c = r \oplus H_I(R, Y)$. He sends the pair (R, c) to Alice as his *challenge*.

2. Alice receives (R, c). She computes $Y' = xR$ and $r' = c \oplus H_I(R', Y')$, and checks that $R = r'P$. If so, she sends Y' as her *response*; otherwise she sends $\perp$.

3. Bob receives Y' from Alice. He checks that $Y' = Y$. If so, he accepts that he's talking to Alice; otherwise he becomes suspicious.

We name this the Wrestlers Identification Protocol in G, *W-ident*^{G} (we drop the superscript to refer to the protocol in general, or when no ambiguity is likely to result). A summary is shown in figure 3.1.

3.2 Security

In order to evaluate the security of our protocol, we present a formal description of the algorithms involved in figure 3.1. Here, the hash function $H_I(\cdot, \cdot)$ is modelled as a random oracle.

Completeness Suppose that Bob really is talking to Alice. Note that $Y' = xR = x(rP) = r(xP) = rX = Y$. Hence $r' = c \oplus H_I(R', Y') = c \oplus H_I(R, Y) = r$, so $r'P = rP = R$, so Alice returns $Y' = Y$ to Bob. Therefore *W-ident* is *complete*: if Bob really is communicating with Alice then he accepts.

Soundness We next show that impersonating Alice is difficult. The natural way to prove this would be to give an adversary a challenge and prove that its probability of giving a correct

$\mathbf{Game}^{\text{imp-}n}_{\textit{W-ident}}(A)$:
  $H_I \xleftarrow{\$} \mathcal{F}[G^2 \to \Sigma^{\ell_I}]$;
  $(x, X) \leftarrow \textit{setup}()$;
  $\textit{win} \leftarrow 0$;
  $\textit{R-map} \leftarrow \varnothing$;
  $\mathbf{c} \leftarrow \textit{challenges}(n)$;
  $(R', Y') \leftarrow A^{H_I(\cdot,\cdot),\textit{check}(\cdot,\cdot)}(X, \mathbf{c})$;
  return $\textit{win}$;

Function $\textit{challenges}(n)$:
  for $i \in \mathbb{N}_{<n}$ **do**
    $(Y, R, c) \leftarrow \textit{challenge}^{H_I(\cdot,\cdot)}$;
    $\textit{R-map} \leftarrow \textit{R-map} \cup \{R \mapsto Y\}$;
    $\mathbf{c}[i] \leftarrow (R, c)$;
  return $\mathbf{c}$;

Function $\textit{check}(R', Y')$:
  if $R' \notin \operatorname{dom} \textit{R-map}$ **then return** 0;
  $Y \leftarrow \textit{R-map}(R')$;
  if $\textit{verify}(Y, Y')$ **then**
    $\textit{win} \leftarrow 1$;
    return 1;
  return 0;

Figure 3.3: Soundness of W-ident: $\mathbf{Game}^{\text{imp-}n}_{\textit{W-ident}}(A)$

response is very small. However, we prove a stronger result: we show that if the adversary can respond correctly to any of a large collection of challenges then it can solve the MCDH problem.

Consider the game $\mathbf{Game}^{\text{imp}}_{\textit{W-ident}}$ shown in figure 3.3. An adversary's probability of successfully impersonating Alice in our protocol, by correctly responding to any one of n challenges, is exactly its probability of winning the game (i.e., causing it to return 1).

3.2.1 Theorem (Soundness of W-*ident*) *Let A be any adversary running in time t and making q_I queries to its random oracle, and q_V queries to its verification oracle. Let G be a cyclic group. Then*

$$\Pr[\mathbf{Game}^{\text{imp-}n}_{\textit{W-ident}^G}(A) = 1] \leqslant \mathbf{InSec}^{\text{mcdh}}(G; t', q_I + q_V)$$

where $t' = t + O(q_I) + O(q_V)$.

3.2.2 Remark Note that the security bound here is *independent* of the value of n. □

Proof We prove this by defining a sequence of games $\mathbf{G}_i$. The first will be the same as the attack game $\mathbf{Game}^{\text{imp-}n}_{\textit{W-ident}}(A)$ and the others will differ from it in minor ways. In each game $\mathbf{G}_i$, let S_i be the event that A wins the game – i.e., that it successfully impersonates the holder of the private key x.

Let game $\mathbf{G}_0$ be the attack game $\mathbf{Game}^{\text{imp}}_{\textit{W-ident}}(A)$, and let (R', Y') be the output of A in the game.

We define a new game $\mathbf{G}_1$ which is the same as $\mathbf{G}_0$, except that we query the random oracle H_I at (R', Y') whenever the adversary queries $\textit{check}(R', Y')$. (We don't charge the adversary for this.) This obviously doesn't affect the adversary's probability of winning, so $\Pr[S_1] = \Pr[S_0]$.

Game $\mathbf{G}_2$ is like $\mathbf{G}_1$, except that we change the way we generate challenges and check their responses. Specifically, we new functions $\textit{challenges}_2$ and $\textit{check}_2$, as shown in figure 3.4.

Function *challenges*$_2(n)$:
  $r^* \xleftarrow{\$} I$; $R^* \leftarrow r^* P$; $Y^* \leftarrow r^* X$;
  for $i \in \mathbb{N}_{<n}$ **do**
    $r \xleftarrow{\$} I$; $R \leftarrow rR^*$; $Y \leftarrow rY^*$;
    $h \leftarrow H_I(R, Y)$; $c \leftarrow r \oplus h$;
    $\mathit{R\text{-}map} \leftarrow \mathit{R\text{-}map} \cup \{R \mapsto r\}$;
    $\mathbf{c}[i] \leftarrow (R, c)$;
  return c;

Function *check*$_2(R', Y')$:
  if $R' \notin \operatorname{dom} \mathit{R\text{-}map}$ **then return** 0;
  $r \leftarrow \mathit{R\text{-}map}(R')$;
  if $\mathit{verify}(Y^*, Y'/r)$ **then**
    $\mathit{win} \leftarrow 1$;
    return 1;
  return 0;

Figure 3.4: Soundness of W-ident: challenges$_2$ and check$_2$

Function *challenges*$_4(n)$:
  $r^* \xleftarrow{\$} I$; $R^* \leftarrow r^* P$; $Y^* \leftarrow r^* X$;
  for $i \in \mathbb{N}_{<n}$ **do**
    $r \xleftarrow{\$} I$; $R \leftarrow rR^*$;
    $c \xleftarrow{\$} \Sigma^{\ell_I}$;
    $\mathit{R\text{-}map} \leftarrow \mathit{R\text{-}map} \cup \{R \mapsto r\}$;
    $\mathbf{c}[i] \leftarrow (R, c)$;
  return c;

Figure 3.5: Soundness of W-ident: challenges$_4$

While we're generating and checking challenges in a more complicated way here, we're not actually changing the distribution of the challenges, or changing the winning condition. Hence $\Pr[S_2] = \Pr[S_1]$.

Now we change the rules again. Let $\mathbf{G}_3$ be the same as $\mathbf{G}_2$ except that we change the winning condition. Instead, we say that the adversary wins if any of the queries to its random oracle $H_I(R', Y')$ would be a correct response – i.e., *check*$_2(R', Y')$ would return 1. Since we query the oracle on (R', Y') on its behalf at the end of the game, no adversary can do worse in this game than it does in $\mathbf{G}_2$, so we have $\Pr[S_3] \geqslant \Pr[S_2]$. (It's not hard to see that this only helps quite stupid adversaries. We can transform any adversary into one for which equality holds here.)

Finally, let $\mathbf{G}_4$ be the same as $\mathbf{G}_3$ except that we change the way we generate challenges again: rather than computing h and setting $c \leftarrow h \oplus r$, we just choose c at random. Specifically, we use the new function, *challenges*$_4$, shown in figure 3.5.

Since $H_I(\cdot, \cdot)$ is a random function, the adversary can only distinguish $\mathbf{G}_4$ from $\mathbf{G}_3$ if it queries its random oracle at some (R, rY^*). But if it does this, then by the rule introduced in $\mathbf{G}_3$ it has already won. Therefore we must have $\Pr[S_4] = \Pr[S_3]$.

Our *challenges*$_4$ function is interesting, since it doesn't actually make use of r^* or Y^* when generating its challenges. This gives us the clue we need to bound $\Pr[S_4]$: we can use adversary A to solve the multiple-guess Diffie-Hellman problem in G by simulating the game

Adversary $B^{V(\cdot)}(X, R^*)$: $\quad F \leftarrow \varnothing$; *R-map* $\leftarrow \varnothing$; $\quad$**for** $i \in \mathbb{N}_{<n}$ **do** $\quad\quad r \xleftarrow{\$} I$; $R \leftarrow rR^*$; $c \xleftarrow{\$} \Sigma^{\ell_I}$; $\quad\quad$*R-map* $\leftarrow$ *R-map* $\cup \{R \mapsto r\}$; $\quad\quad \mathbf{c}[i] \leftarrow (R, c)$; $\quad (R', Y') \leftarrow A^{H_I(\cdot,\cdot),\mathit{check}(\cdot,\cdot)}(X, \mathbf{c})$; $\quad$**if** $Y' \neq \perp$ **then** $H_I(R', Y')$;	Oracle $H_I(R', Y')$: $\quad$**if** $(R', Y') \in \operatorname{dom} F$ **then** $\quad\quad h \leftarrow F(x)$; $\quad$**else** $\quad\quad$*check*(R', Y'); $\quad\quad h \xleftarrow{\$} \Sigma^{\ell_I}$; $F \leftarrow F \cup \{(R', Y') \mapsto h\}$; $\quad$**return** h; Oracle *check*(R', Y'): $\quad$**if** $R' \in \operatorname{dom}$ *R-map* **then** $V(Y'/\mathit{R\text{-}map}(R'))$;

Figure 3.6: Soundness of W-ident: reduction from MCDH

$\mathbf{G}_4$. Specifically, we define the adversary B as shown in figure 3.6. That is, for each query A makes to its random oracle at a new pair (R', Y'), we see whether this gives us the answer we're looking for. We have $\Pr[S_0] \leqslant \Pr[S_4] = \mathbf{Succ}^{\text{mcdh}}_G(B) \leqslant \mathbf{InSec}^{\text{gdh}}(G; t', q_I + q_V)$ as required. □

Zero-knowledge Finally we must prove that W-*ident* is (statistical) zero-knowledge – i.e., that, except with very small probability, Bob learns nothing of use to him except that he's interacting with Alice. To do this, we show that, for any algorithm B which Bob might use to produce his challenge to the real Alice, there exists a simulator S which produces transcripts distributed very similarly to transcripts of real conversations between B and Alice, the difference being that S doesn't know Alice's key. We shall show that the statistical difference between the two distributions is $2^{-\ell_I}$.

The intuition here is that Bob ought to know what answer Alice is going to give him when he constructs his challenge. This is certainly true if he's honest: his challenge is $R = rP$ for some r he knows, so he won't learn anything useful when Alice responds with $xR = rX$. However, if Bob sends a challenge R when he doesn't know the corresponding r, he learns something potentially useful. The accompanying check value $c = r \oplus H_I(R, rX)$ keeps him honest.

To show this, we present an *extractor* which, given any challenge (R, c) Bob can construct, and his history of random-oracle queries, either returns a pair (r, Y) such that $R = rP$ and $Y = rX$, or $\perp$; moreover, the probability that Alice returns a response $Y' \neq \perp$ given the challenge (R, c) is $2^{-\ell}$. We can, of course, readily convert this extractor into a simulator to prove the zero-knowledge property of our protocol.

We shall actually consider a slightly more complex setting. We grant Bob access to an oracle which produces random, correctly-formed challenges. We require this to model the legitimate challenges of other parties when we analyse the security of our key exchange protocol. The extractor is permitted to fail given

3.2.3 Definition (Discrete-log extractors) Let T, B be randomized algorithms. Define the game $\mathbf{Game}^{\text{dl-ext}}_G(T, B)$ as shown in figure 3.7. The *success probability of T as a discrete-log extractor against B* is defined as

$$\mathbf{Succ}^{\text{dl-ext}}_G(T, B) = \Pr[\mathbf{Game}^{\text{dl-ext}}_G(T, B) = 1].$$

□

$\textbf{Game}^{\text{dl-ext}}_G(T, B)$:
$H_I \xleftarrow{\$} \mathcal{F}[G^2 \to \Sigma^{\ell_I}]$; $Q_H \leftarrow \varnothing$; $Q_C \leftarrow \varnothing$;
$(x, X) \leftarrow \mathit{setup}()$;
$(R, c) \leftarrow B^{H_I\text{-}\mathit{trap}(\cdot,\cdot), C()}(x, X)$;
$(r, Y) \leftarrow T(R, c, Q_H)$;
$Y' \leftarrow xR$; $h' \leftarrow H_I(R, Y')$; $r' \leftarrow c \oplus h'$;
if $r \neq \perp$ **then**
  if $Y = \perp \vee R \neq rP \vee Y \neq Y'$ **then return** 0;
if $R = r'P$ **then** $(r^*, Y^*) = (r', Y')$;
else $(r^*, Y^*) \leftarrow (\perp, \perp)$;
if $(R, c) \in Q_C$ **then return** 1;
if $(r, Y) = (r', Y')$ **then return** 1;
return 0;

Oracle H_I-*trap*(R', Y'):
$h \leftarrow H_I(R', Y')$;
$Q_H \leftarrow Q_H \cup \{(R', Y', h)\}$;
return h;

Oracle $C()$:
$r \xleftarrow{\$} \mathbb{N}_{<q}$;
$R \leftarrow rP$; $c \leftarrow r \oplus H_I(R, rX)$;
$Q_C \leftarrow Q_C \cup \{(R, c)\}$;
return (R, c)

Figure 3.7: Discrete log extraction game: $\textbf{Game}^{\text{dl-ext}}_G(T, B)$

Let's unpack this definition slightly. We make the following demands of our extractor.

- It is given a bare 'transcript' of B's execution. In particular, it is given only its output and a list of B's random-oracle queries in no particular order.
- While the extractor is not given the private key x, the adversary B is given the private key.
- We require that, if the extractor produces values $r, Y \neq \perp$ then r and Y are *correct*; i.e., that $R = rP$ and $Y = xR$.
- The extractor is explicitly *not* given the outputs of the challenge-generation oracle $C()$, nor of the random-oracle queries issued by $C()$. However, we allow the extractor to fail (i.e., return $\perp$) if B simply parrots one of its C-outputs.
- The extractor is allowed – indeed *required* – to fail if the challenge (R, c) is *invalid* (i.e., Alice would return $\perp$ given the challenge).

The resulting definition bears a striking similarity to the concept of *plaintext awareness* in [BDPR98].

Such an extractor indeed exists, as the following lemma states.

3.2.4 Lemma (Effectiveness of extractor $T_{W\text{-}ident}$) *There exists a* universal discrete-log extractor $T_{W\text{-}ident}$ *which, for any algorithm B,*

$$\textbf{Succ}^{\text{dl-ext}}_G(T_{W\text{-}ident}, B) \geqslant 1 - \frac{1}{2^{\ell_I}}.$$

Moreover, if B issues at most q_H random-oracle queries, then the running time of $T_{W\text{-}ident}$ is $O(q_H)$.

We prove this result at the end of the section. For now, let us see how to prove that W-*ident* is zero-knowledge.

We use the set-up described in section 2.10. Our initialization function $I_{W\text{-}ident}$ just chooses a random $x \in \mathbb{N}_{<q}$ as the world private input and sets $X = xP$ as the common input. In the

Initialization function $I_{W\text{-}ident}()$:
  $x \xleftarrow{\$} \mathbb{N}_{<|G|}$;
  $X \leftarrow xP$;
  return (x, X);

Real-prover world $W^{H_I(\cdot,\cdot)}_{W\text{-}ident}((x, X), \sigma, \tau, \mu)$:
  if $\tau =$ challenge **then**
    $(R, c) \leftarrow \mu$;
    $Y \leftarrow response^{H_I(\cdot,\cdot)}(R, c, x)$;
    return $(1, Y)$;
  else
    return $(\sigma, \perp)$;

Simulator $S_{W\text{-}ident}$'s fake world $W^{H_I(\cdot,\cdot)}_{\text{sim}}((X, u), \sigma, \tau, \mu)$:
  if $\tau =$ init **then**
    return $(\varnothing, \lambda)$;
  $Q_H \leftarrow \sigma$;
  if $\tau =$ challenge **then**
    $(R, c) \leftarrow \mu$;
    $(r, Y) \leftarrow T_{W\text{-}ident}(R, c, Q_H)$;
    return (Q_H, Y);
  else if $\tau =$ random **then**
    $(i, (R', Y'), h) \leftarrow \mu$;
    $Q_H \leftarrow Q_H \cup \{(R', Y', h)\}$;
    return (Q_H, λ);
  else
    return $(\sigma, \perp)$;

Figure 3.8: Real-prover and simulator for zero-knowledge of W-ident

'real world', the adversary is allowed to submit a (single) challenge to the prover; it is given the prover's response, and must then compute its output. This is shown on the left hand side of figure 3.8.

The zero-knowledge property of the scheme is described by the following theorem.

3.2.5 Theorem (Statistical zero-knowledge of W-*ident*) *Let $I_{W\text{-}ident}$, $W_{W\text{-}ident}$ and $S_{W\text{-}ident}$ be the real-prover world and simulator shown in figure 3.8. Then, for any t, q_I and q_C,*

$$\mathbf{InSec}^{\text{sim}}(W_{W\text{-}ident}, I_{W\text{-}ident}, S_{W\text{-}ident}; t, q_I, q_C, 0) \leqslant \frac{q_C}{2^{\ell}_I}.$$

where q_C is the maximum number of challenges allowed by the adversary.

Proof The simulator simply uses the extractor $T_{W\text{-}ident}$ to extract the answer from the adversary's history of random oracle queries. Observe that $S_{W\text{-}ident}$ is a fake-world simulator. By lemma 3.2.4, the extractor fails with probability only $2^{-\ell_I}$. The theorem follows by a simple union bound and proposition 2.10.4. □

We now return to proving that our extractor exists and functions as claimed. The following two trivial lemmata will be useful, both now and later.

3.2.6 Lemma (Uniqueness of discrete-logs) *Let $G = \langle P \rangle$ be a cyclic group. For any $X \in G$ there is a unique integer x where $0 \leqslant x < |G|$ and $X = xP$.*

Proof Certainly such an x exists, since G is cyclic and finite. Suppose $X = xP = x'P$: then $0 = xP - x'P = (x - x')P$. Hence $(x - x')$ is a multiple of $|G|$. □

3.2.7 Lemma (Uniqueness of check values) *Let $G = \langle P \rangle$ be a cyclic group; let H_I be a function $H_I \colon \Sigma^{2\ell_G} \to \Sigma^{\ell_I}$. Fix some $x \in \mathbb{N}_{<|G|}$ and define the set*

$$V_x = \bigl\{\, (R, c) \in G \times \Sigma^{\ell_I} \bigm| R = \bigl(c \oplus H_I(R, xR)\bigr)P \,\bigr\}.$$

Extractor $T_{W\text{-}ident}(R, c, Q_H)$:
  for (R', Y', h) **in** Q_H **do**
    $r \leftarrow h \oplus c$;
    if $R = R' = rP \wedge Y' = rX$ **then return** (r, Y');
  return $(\perp, \perp)$;

Figure 3.9: The discrete-log extractor $T_{W\text{-}ident}$

Then, for any R, c, c', if $(R, c) \in V_x$ and $(R, c') \in V_x$ then $c = c'$.

Proof From lemma 3.2.6, we see that there is a unique $r \in \mathbb{N}_{<|G|}$ for which $R = rP$. Now, if $(R, c) \in V_x$, we must have $r = c \oplus H_I(R, xR)$. It follows that $c = r \oplus H_I(R, xR)$. □

Proof of lemma 3.2.4 Our extractor $T_{W\text{-}ident}$ is shown in figure 3.9.

Let B be any randomized algorithm, and let (R, c, Q_H) be as given to the extractor by $\mathbf{Game}_G^{\text{dl-ext}}(T_{W\text{-}ident}, B)$. Let the quantities H_I, Q_C, r, r', x and X be as in that game.

Suppose that the extractor returns values $(r, Y) \neq (\perp, \perp)$. Let $h = r \oplus c$; then there must be a query $(R, Y, h) \in Q_H$, and we have $R = rP$ and $Y = rX = r(xP) = x(rP) = xR = Y'$, so the extractor's output must be correct unless it fails.

Furthermore, in the case where the extractor did not fail, we have $h = H_I(R, Y) = H_I(R, Y')$ and $c = r \oplus h$, so the challenge was valid. Therefore, if the challenge was invalid, the extractor will fail.

We now deal with the challenge-generation oracle. Suppose that $(R, c') \in Q_C$ for some c'. Now, if $c = c'$ then (R, c') is a repeat of some challenge from the challenge-generation oracle, and the extractor is permitted to fail. On the other hand, suppose $c \neq c'$; then, the challenge (R, c) must be invalid by lemma 3.2.7, so the extractor is required to fail, and we have established that indeed it will. From now on, suppose that R is distinct from all the R-values returned by $C()$.

Let $Y = xR$. Suppose that B queried its random oracle at (R, Y). Let $h = H_I(Y)$, so $r' = c \oplus h$. If the challenge is valid then $R = r'P$; therefore $Y = xR = xr'P = r'X$, so we have $(R, Y, h) \in Q_H$ with $R = rP$ and $Y = rX$. Hence the extractor returns $r = r'$ as required.

It remains to deal with the case where there is no random-oracle query at (R, Y). But then $h = H_I(R, Y)$ is uniformly distributed, and independent of the entire game up to this point. Let r be the correct discrete log of R; by lemma 3.2.6 there is only one possible value. The extractor always fails under these circumstances, but a correct responder would reply with probability

$$\Pr[h = c \oplus r] = \frac{1}{2^{\ell_I}}.$$

This concludes the proof. □

3.2.8 **Remark** Note that the fact that the algorithm B was given the private key is irrelevant to the above argument. However, we shall need this property when we come to prove deniability for the key-exchange protocol. □

3.2.9 Remark It's easy to see from the above proof that the extractor works flawlessly on the 'honest verifier' algorithm *challenge* shown in figure 3.2. This shows that W-*ident* is perfect zero-knowledge against honest verifiers. We're much more interested in dishonest verifiers, though. □

3.3 An identity-based identification scheme

Boneh and Franklin [BF03] showed how to construct an identity-based encryption scheme using bilinear pairings. The resulting encryption scheme looks somewhat like a pairing-based version of ElGamal's encryption scheme [ElG85]. We can easily apply their techniques to our identification protocol, and thereby obtain an identity-based identification scheme. Providing the necessary formalisms to prove theorems analogous to our theorems 3.2.1 and 3.2.5 would take us too far from our objectives; but given appropriate security notions, we can readily adapt our existing proofs to the new setting.

Bilinear pairings Before we describe the necessary modifications to the protocol, we first give a (very brief!) summary of cryptographic pairings. (The Boneh-Franklin paper [BF03] gives more detail; also [Men05] provides a useful introduction to the topic.)

Let $(G, +)$, $(G', +)$ and $(G_T, \times)$ be cyclic groups with prime order q; let $P \in G$ and $P' \in G'$ be elements of order q in G and G' respectively. We say that a mapping $\hat{e}\colon G \times G' \to G_T$ is a *non-degenerate bilinear pairing* if it satisfies the following properties.

Bilinearity For all $R \in G$ and $S', T' \in G'$, we have $\hat{e}(R, S' + T') = \hat{e}(R, S')\,\hat{e}(R, T')$; and for all $R, S \in G$ and $T' \in G'$ we have $\hat{e}(R + S, T') = \hat{e}(R, T')\,\hat{e}(S, T')$.

Non-degeneracy $\hat{e}(P, P') \neq 1$.

For practical use, we also want $\hat{e}(\cdot, \cdot)$ to be efficient to compute. The reader can verify that $\hat{e}(aP, bP') = \hat{e}(P, P')^{ab}$. It is permitted for the two groups G and G' to be equal.

We require a different intractability assumption, specifically that the *bilinear* Diffie-Hellman problem (BDH) – given $(aP, bP, aP', cP') \in G^2 \times G'^2$, find $\hat{e}(P, P')^{abc} \in G_T$ – is difficult. This implies the difficulty of the computational Diffie-Hellman problem in all three of G, G' and G_T.

The identity-based scheme We need a trusted authority; following [Sch96] we shall call him Trent. Trent's private key is $t \in \mathbb{N}_{<q}$; his public key is $T = tP$.

Finally, we need cryptographic hash functions $H_I\colon G \times G_T \to \Sigma^{\ell_I}$ and $H_{ID}\colon \Sigma^* \to G'$; a formal security analysis would model these as random oracles.

Alice's public key is $A = H_{ID}(\texttt{Alice}) \in G'$. Her private key is $K_A = tA \in G'$ – she needs Trent to give this to her. Bob can interact with Alice in order to verify her identity as follows.

1. Bob computes $\gamma_A = \hat{e}(T, A) \in G_T$. (He can do this once and store the result if he wants, but it's not that onerous to work it out each time.)
2. Bob chooses $r \in_{\$} \mathbb{N}_{<q}$, and sets $R = rP$. He also computes $\psi = \gamma_A^r$, $h = H_I(R, \psi)$ and $c = r \oplus h$. He sends his challenge (R, c) to Alice.
3. Alice receives (R', c'). She computes $\psi' = \hat{e}(R, K_A)$, $h' = H_I(R', \psi')$, and $r' = c' \oplus h')$. She checks that $R' = r'P$; if so, she sends ψ' back to Bob; otherwise she refuses to talk to him.

4. Bob receives ψ'. If $\psi = \psi'$ then he accepts that he's talking to Alice.

This works because $\psi = \gamma_A^r = \hat{e}(T, A)^r = \hat{e}(tP, A)^r = \hat{e}(rP, A)^t = \hat{e}(R, tA) = \psi'$.

Informal analysis An analogue to lemma 3.2.4 can be proven to show how to extract r from a verifier's random-oracle queries; statistical zero knowledge would then follow easily, as in theorem 3.2.5. Soundness is intuitively clear, since an adversary must compute $\psi = \hat{e}(P, P')^{art}$ given $A = aP'$, $R = rP$ and $T = tP$, which is an instance of the BDH problem. An analogue of theorem 3.2.1 would have to prove this for an adversary capable of making identity requests as well as obtaining challenges. Finally, our key-exchange protocol can be constructed out of this identity-based identification scheme, yielding an identity-based authenticated key-exchange protocol. We leave it to the reader to work through the details.

3.4 Comparison with the protocol of Stinson and Wu

Our protocol is similar to a recent proposal by Stinson and Wu [SW06]. They restrict their attention to Schnorr groups $G \subset \mathbb{F}_p^*$. Let γ be an element of order $q = |G|$. The prover's private key is $a \in_\$ \mathbb{N}_{<q}$ and her public key is $\alpha = \gamma^a$. In their protocol, the challenger chooses $r \in_\$ \mathbb{N}_{<q}$, computes $\rho = \gamma^r$ and $\psi = \alpha^r$, and sends a challenge $(\rho, H(\rho, \psi))$. The prover checks that $\rho^q \neq 1$, computes $\psi = \rho^a$, checks the hash, and sends ψ back by way of response. They prove their protocol's security in the random-oracle model.

Both the Wrestlers protocol and Stinson-Wu require both prover and verifier to compute two exponentiations (or scalar multiplications) each. The sizes of the messages used by the two protocols are also identical.

(An earlier version of the Stinson-Wu protocol used a cofactor exponentiation: if we set $f = (p-1)/q$, then we use $\psi = \alpha^{rf}) = \rho^{af} = \gamma^{afr}$. This is more efficient in typical elliptic curve subgroups, since the cofactor of such subgroups is usually small: indeed, [Cer00] recommends rejecting groups with cofactor $f > 4$. However, in the Schnorr groups used by Stinson and Wu, the cofactor is much larger than q, and their new variant is more efficient.)

We note that the zero-knowledge property of the Stinson-Wu protocol requires the Diffie-Hellman knowledge of exponent assumption (KEA). Very briefly: suppose A is a randomized algorithm which takes as input $X \in G$ and outputs a pair (rP, rX); intuitively, the KEA asserts A must have done this by choosing r somehow and then computing its output from it. Formally, it asserts the existence of an 'extractor' algorithm which takes as input the element X and the random coins used by A and outputs r with high probability. This is a very strong assumption, and one which is unnecessary for our protocol, since we can present an *explicit* extractor.

The KEA assumption as stated in [SW06] allows the extractor to fail with some negligible probability, over and above the probability that a dishonest verifier managed to guess the correct $h = H(\rho, \psi)$ without making this random-oracle query. Not only does our protocol achieve zero- knowledge without the KEA, our extractor is, in this sense, 'perfect'.

Our protocol is just as strong as Stinson-Wu under attack from active intruders: see table 3.1 for a very brief sketch of the case-analysis which would be the basis of a proof of this.

An identity-based analogue of Stinson-Wu can be defined using a bilinear pairing, just as we did in section 3.3. However, to prove the zero-knowledge property, one needs to make a bilinear analogue of the knowledge of exponent assumption.

Challenge		Response	Security
R	c	Y	Nothing to prove.
R	c'	—	Prover rejects by lemma 3.2.7; Y' probably wrong by theorem 3.2.1.
R	c	Y'	Response is incorrect.
R'	—	Y	Response is incorrect.
R'	c	Y'	Prover rejects with probability $1 - 2^{-\ell_I}$; Y' probably wrong by theorem 3.2.1.
R'	c'	Y'	Simulate prover using extractor (lemma 3.2.4); Y' probably wrong by theorem 3.2.1.

Table 3.1: Security of W-ident against active intruders (summary)

We suspect that a key-exchange protocol like ours can be constructed using Stinson-Wu rather than the Wrestlers identification scheme. We haven't, however, gone through all the details, since we believe our protocol is just as efficient and is based on much more conservative assumptions.

4 A simple key-exchange protocol

In this section, we describe a simple key-exchange protocol built out of the identification protocol shown previously.

The key-exchange protocol arises from the following observation. If Bob sends a challenge, presumably to Alice, and gets a correct response, then not only did he really send the challenge to Alice but he knows that she received it correctly.

So, if Alice and Bob authenticate each other, by the end of it, they should each have chosen a random private value, sent the corresponding public value to the other, and been convinced that it arrived safely.

Unfortunately, life isn't quite this kind, and we have to do some more work to make this scheme secure.

Our key exchange protocol essentially consists of two parallel instances of the identification protocol. If Alice receives a correct response to her challenge, she will know that Bob received her challenge correctly, and *vice versa*. If we let Alice's challenge be $R_0 = r_0 P$ and Bob's challenge be $R_1 = r_1 P$ then each can compute a shared secret $Z = r_0 R_1 = r_0 r_1 P = r_1 R_0$ unknown to an adversary. There are, unfortunately, a few subtleties involved in turning this intuition into a secure key-exchange protocol, which we now describe.

4.1 Overview

We present a quick, informal description of our basic key-exchange protocol. In addition to our group G, we shall also need a secure symmetric encryption scheme $\mathcal{E} = (\kappa, E, D)$, and two secure hash functions $H_I \colon \Sigma^{2\ell_G} \to \Sigma^{\ell_I}$ and $H_K \colon \Sigma^{\ell_G + 1} \to \Sigma^{\kappa}$.

Setup	Group $G = \langle P \rangle$; $\lvert G \rvert = q$ is prime. $H_I(\cdot,\cdot)$ and $H_K(\cdot)$ are secure hashes. $\mathcal{E} = (\kappa, E, D)$ is an IND-CCA2 symmetric encryption scheme.
Parties	U_i for $0 \leqslant i < n$.
Private keys	$x_i \in_\$ \mathbb{N}_{<q}$.
Public keys	$X_i = x_i P$.

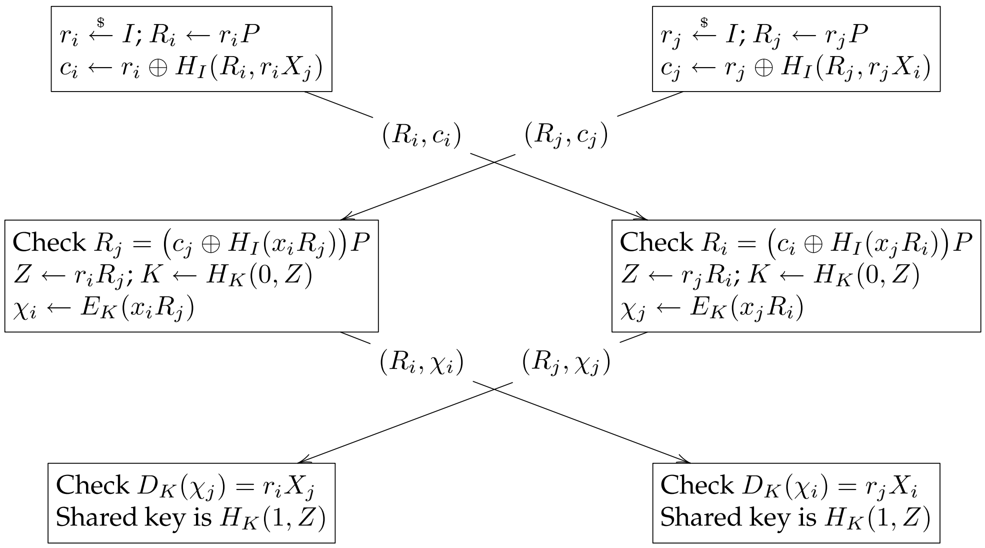

Figure 4.1: Summary of the Wrestlers Key Exchange protocol, W-kx

Suppose that Alice's and Bob's private keys are a and b respectively, and their public keys are $A = aP$ and $B = bP$.

1. Alice chooses a random index $r \in_\$ \mathbb{N}_{<\lvert G \rvert}$. She computes $R = rP$ and $c = r \oplus H_I(R, rB)$. She sends the pair (R, c) to Bob.

2. Similarly, Bob chooses a random $s \in_\$ \mathbb{N}_{<\lvert G \rvert}$. He computes $S = sP$ and $d = s \oplus H_I(S, sA)$. He sends (S, d) to Alice.

3. Alice receives (S', d') from Bob. She computes $s' = d' \oplus H_I(S', aS')$, and verifies that $S' = s'P$. If so, she computes $K_A = H_K(0 \parallel rS')$, and sends $R, E_{K_A}(aS')$ to Bob.

4. Similarly, Bob receives (R', c') from Alice. He verifies that $R' = \big(c' \oplus H_I(R', bR')\big)P$. If so, he computes $K_B = H_K(0 \parallel sR')$ and sends S, $E_{K_B}(bR')$ to Alice.

5. Alice receives a ciphertext (S'', χ_B) from Bob. She checks that $S'' = S'$, decrypts χ_B, and checks that $D_{K_A}(\chi_B) = rB$. If so, she uses $H_K(1 \parallel rS')$ as her shared secret.

6. Similarly, Bob receives (R'', χ_A) from Alice, and checks $R'' = R'$ and $D_{K_B}(\chi_A) = sA$. If so, he uses $H_K(1 \parallel sR')$ as his shared secret.

This is the Wrestlers Key Exchange protocol, $W\text{-}kx^{G,\mathcal{E}}$ (again, we omit the superscripts when referring to the general protocol, or when confusion is unlikely). A diagrammatic summary of the protocol is shown in figure 4.1.

Assume, for the moment, that Alice and Bob's messages are relayed honestly. Then:

- $aS' = aS = a(sP) = s(aP) = sA$, so $s' = d' \oplus H_I(S' aS') = d \oplus H_I(S, sA) = s$, and $S' = S = sP = s'P$, and therefore Alice responds to Bob's message;
- similarly $bR' = rB$, so $r' = r$ and $R' = r'P$, and therefore Bob responds to Alice's message;
- $bR' = bR = b(rP) = r(bP) = rB$, and $aS' = aS = a(sP) = s(aP) = sA$, and therefore both parties compute their responses correctly; and
- $rS' = rS = r(sP) = s(rP) = sR = sR'$, so $K_A = K_B$, and therefore they can decrypt each others' responses, and agree the same shared secret.

This shows that the protocol is basically valid, but says little about its security. The remainder of this section will describe our protocol in more formal detail, and prove its security in a model with multiple parties and an adversary who controls the network.

Observe that the protocol as we've presented here is *symmetrical*. There's no notion of 'initiator' or 'responder'. There are a total of four messages which must be sent before both parties accept. However, this can be reduced to three by breaking the symmetry of the protocol and combining one or other party's challenge and response messages. We choose to analyse the symmetrical version, since to do so, it suffices to consider only the two different kinds of messages. Since our security model allows the messages to be adversarially delayed and reordered, it is easy to show that the security of an optimized, asymmetrical protocol is no worse than the symmetrical version we present here.

4.2 Security model and security definition

Our model is very similar to that of Canetti and Krawczyk [CK01], though we have modified it in two ways.

1. We allow all the participants (including the adversary) in the protocol access to the various random oracles required to implement it.
2. Since we want to analyse a specific, practical scheme, asymptotic results are useless. We measure the adversary's resource usage carefully, and produce a quantitative bound on the adversary's advantage in the SK-security game.

Overview We briefly describe our modified model, pointing out the changes we have made, and how they apply to our protocol. Much of Canetti and Krawczyk's model (for example, the local and global outputs) is useful for proving more general security properties such as demonstrating that SK-security suffices for constructing secure channels, and we shall not concern ourselves with such details. Other parts deal with issues such as security parameters and ensuring that all the computation is polynomially bounded, which are irrelevant since we are dealing with a single concrete protocol rather than a family of them.

The entities in the model are the *adversary* A, and a (fixed) number of *parties* P_i, for $0 \leqslant i < n$. If the protocol under consideration makes use of random oracles, then all the participants – the adversary and the parties – are all allowed access to the random oracles.

The parties and the adversary play a 'game'. At the beginning of the game, the participants are given some inputs computed by a randomized *initialization procedure init*. This produces as output a pair $(i_U, \mathbf{i})$; the value i_U is the *global input*, and is given to all the participants including the adversary. The vector $\mathbf{i}$ has n components, and party P_i is given $(i_U, \mathbf{i}[i])$ as input.

Sessions Parties don't act directly. Instead, each party runs a number of *sessions*. A session a triple $S = (P_i, P_j, s)$, where $i, j \in \mathbb{N}_{<n}$ identify the owning party and a *partner*, and $s \in \Sigma^{\ell_S}$ is a *session-id*. (The original model includes a rôle, for distinguishing between initiators and responders. Our protocol is symmetrical, so this distinction isn't useful.) If P_i runs a session $S = (P_i, P_j, s)$ and P_j runs a session $S' = (P_j, P_i, s)$ then we say that S and S' are *matching*, and that P_j is P_i's *partner* for the session.

At most one participant in the game is *active* at any given time. Initially the adversary is active. The adversary may *activate* a session in one of two ways.

1. It may *create a session* of a party P_i, by selecting a session-id $s \in \Sigma^{\ell_S}$ and a partner j. There is no requirement that P_j ever have a matching session. However, all sessions of a party must be distinct, i.e., sessions with the same partner must have different session-ids.
2. It may *deliver a message* $\mu \in \Sigma^*$, from party P_j, to an existing session $S = (P_i, P_j, s)$. There is no requirement that any party previously sent μ: the adversary is free to make up messages as it sees fit.

The adversary becomes inactive, and the session becomes active. The session performs some computation, according to its protocol, and may request a message μ be delivered to the matching session running in its partner (which may not exist). The session may also *terminate*. In the case we are interested in, of key-exchange protocols, a session $S = (P_i, P_j, s)$ may terminate in one of two ways:

1. it may *complete*, outputting (i, j, s, K), for some *session key* K, or
2. it may *abort*, outputting $(i, j, s, \perp)$.

Once it has performed these actions, the session deactivates and the adversary becomes active again. The adversary is given the message μ, if any, and informed of whether the session completed or aborted, but, in the case of completion, not of the value of the key K. A session is *running* if it has been created and has not yet terminated.

Other adversarial actions As well as activating sessions, the adversary has other capabilities, as follows.

- It may *expire* any session S, causing the owning party to 'forget' the session key output by that session.
- It may *corrupt* any party P_i, at will: the adversary learns the entire state of the corrupted party, including its initial input $\mathbf{i}[i]$, the state of any sessions it was running at the time, and the session keys of any completed but unexpired sessions. Once corrupted, a party can no longer be activated. Of course, the adversary can continue to send messages allegedly from the corrupted party.
- It may *reveal the state* of a running session S, learning any interesting values specific to that session, but *not* the owning party's long-term secrets.
- It may *reveal the session-key* of a completed session S.
- It may elect to be *challenged* with a completed session S, provided. Challenge sessions form part of the security notion for key-exchange protocols. See below for more details.

We say that a session $S = (P_i, P_j, s)$ is *locally exposed* if

- it has had its state revealed,
- it has had its session-key revealed, or
- P_i has been corrupted, and S had not been expired when this happened.

A session is *exposed* if it is locally exposed, or if its matching session exists and has been locally exposed.

At the beginning of the game, a bit b^* is chosen at random. The adversary may choose to be *challenged* with any completed, unexposed session;[2] the adversary is then given either the session's key – if $b^* = 1$ – or a string chosen at random and independently of the game so far from a protocol-specific distribution – if $b^* = 0$. At the end of the game, the adversary outputs a single bit b.

SK-security We've now described the game; it is time to explain the adversary's goal in it. The adversary *wins* the game if either

1. two unexposed, matching sessions complete, but output different keys,[3] or
2. the adversary correctly guesses the hidden bit b^*.

More formally, we make the following definition.

4.2.1 Definition (SK-security) Let $\Pi^{H_0(\cdot), H_1(\cdot), \ldots}$ be a key-exchange protocol which makes use of random oracles $H_0(\cdot)$, $H_1(\cdot)$, $\ldots$, and let A be an adversary playing the game described previously, where n parties run the protocol Π. Let V be the event that any pair of matching, unexposed sessions completed, but output different session keys. Let W be the event that the adversary's output bit matches the game's hidden bit b^*. We define the adversary's *advantage against the SK-security of the protocol* Π to be

$$\mathbf{Adv}^{\mathrm{sk}}_{\Pi}(A, n) = \max(\Pr[V], 2\Pr[W] - 1).$$

Furthermore, we define the *SK insecurity function of the protocol* Π to be

$$\mathbf{InSec}^{\mathrm{sk}}(\Pi; t, n, q_S, q_M, q_{H_0}, q_{H_1}, \ldots) = \max_A \mathbf{Adv}^{\mathrm{sk}}_{\Pi}(A, n)$$

where the maximum is taken over all adversaries A with total running time t (not including time taken by the parties), create at most q_S sessions, deliver at most q_M messages, and (if applicable) make at most q_{H_i} random-oracle queries to each random oracle $H_i(\cdot)$. □

4.3 Security

In order to analyse our protocol $W\text{-}kx^{G,\mathcal{E}}$ properly, we must describe exactly how it fits into the formal model described in our formal model.

[2] The original Canetti-Krawczyk definition restricts the adversary to a single challenge session, but our proof works independent of the number of challenge sessions, so we get a stronger result by relaxing the requirement here.)

[3] The original Canetti-Krawczyk definition differs slightly here. It requires that 'if two *uncorrupted* parties complete matching sessions then they both output the same key' [original emphasis]. This can't be taken at face value, since none of the protocols they claim to be secure actually meet this requirement: they meet only the weaker requirement that parties completing matching sessions output different keys with negligible probability. We assume here that this is what they meant.

Sessions and session-ids Our formal model introduced the concept of sessions, which the informal description of section 4.1 neglected to do. (One could argue that we described a single session only.) As we shall show in section 4.4, our protocol is *insecure* unless we carefully modify it to distinguish between multiple sessions.

In the model, distinct key-exchange sessions are given distinct partners and session-ids. In order to prevent sessions interfering with each other, we shall make explicit use of the session-ids.

Suppose the session-ids are ℓ_S-bit strings. We expand the domain of the random oracle H_I so that it's now

$$H_I\colon G \times \Sigma^{\ell_S} \times G \times G \to \Sigma_{\ell_I}.$$

Messages We split the messages our protocols into two parts: a *type* τ and a *body* μ. We assume some convenient, unambiguous encoding of pairs (τ, μ) as bit-strings. For readability, we present message types as text strings, e.g., '$\mathsf{challenge}$', though in practice one could use numerical codes instead.

The message body itself may be a tuple of values, which, again, we assume are encoded as bit-strings in some convenient and unambiguous fashion. We shall abuse the notation for the sake of readability by dropping a layer of nesting in this case: for example, we write $(\mathsf{hello}, x, y, z)$ rather than $\bigl(\mathsf{hello}, (x, y, z)\bigr)$.

The protocol Our protocol is represented by three functions, shown in figure 4.2.

- $\mathit{init}(n)$ is the initialization function, as described in section 4.2. It outputs a pair $(\mathbf{p}, \mathbf{i})$, where $\mathbf{i}[i]$ is the private key of party P_i and $\mathbf{p}[i]$ is the corresponding public key. Only P_i is given $\mathbf{i}[i]$, whereas all parties and the adversary are given $\mathbf{p}$.
- $\mathit{new\text{-}session}^{H_I(\cdot,\cdot,\cdot,\cdot), H_K(\cdot)}(\mathbf{p}, x, i, j, s)$ is the new-session function. This is executed by party P_i when the adversary decides to create a new session $S = (P_i, P_j, s)$. It is also given the relevant outputs of *init*, and allowed access to the random oracles H_I and H_K.
- $\mathit{message}^{H_I(\cdot,\cdot,\cdot,\cdot), H_K(\cdot)}(\tau, \mu)$ is the incoming-message function. This is executed by a session when the adversary decides to deliver a message (τ, μ) to it. It makes use of the subsidiary functions *msg-challenge* and *msg-response* to handle the messages.

We observe that the protocol never aborts. We could make it do so if it receives an invalid message, but we prefer to ignore invalid messages for the sake of robustness.[4]

Session states We must specify what the adversary obtains when it chooses to reveal a session's state. Given the program in figure 4.2, we can see that the session state consists of the variables (x, X, X', r, R, Y, C).

However, x is the owning party's long-term secret, and it seems unreasonable to disclose this to an adversary who stops short of total corruption.

The public keys X and X' are part of the adversary's input anyway, so revealing them doesn't help. Similarly, the set C of valid challenges could have been computed easily by

[4] Note that this protocol would therefore require modification before it was acceptable in the simulation-based model of [Sho99]. There it is required that a key-exchange protocol terminate after a polynomially-bounded number of messages are delivered to it.

Function *init*(n):
  for $i \in \mathbb{N}_{<n}$ **do**
    $x \xleftarrow{\$} \mathbb{N}_{<|G|}$;
    $\mathbf{i}[i] \leftarrow x$;
    $\mathbf{p}[i] \leftarrow xP$;
  return $(\mathbf{p}, \mathbf{i})$;

Function *new-session*$^{H_I(\cdot,\cdot,\cdot,\cdot), H_K(\cdot)}(\mathbf{p}, x, i, j, s)$:
  $X \leftarrow \mathbf{p}[i]$; $X' \leftarrow \mathbf{p}[j]$; $C \leftarrow \varnothing$;
  $r \xleftarrow{\$} \mathbb{N}_{<|G|}$; $R \leftarrow rP$; $Y \leftarrow rX'$;
  $h \leftarrow H_I(X, s, R, Y)$; $c \leftarrow r \oplus h$;
  send $(\textsf{challenge}, R, c)$;

Function *message*$^{H_I(\cdot,\cdot,\cdot,\cdot), H_K(\cdot)}(\tau, \mu)$:
  if $\tau = \textsf{challenge}$ **then** *msg-challenge*(μ);
  else if $\tau = \textsf{response}$ **then** *msg-response*(μ);

Function *msg-challenge*(μ):
  $(R', c') \leftarrow \mu$;
  $Y' \leftarrow xR'$;
  $h' \leftarrow H_I(X', s, R', Y')$;
  $r' \leftarrow c' \oplus h'$;
  if $R' \neq r'P$ **then return**;
  $C \leftarrow C \cup \{R\}$;
  $Z \leftarrow rR'$;
  $(K_0, K_1) \leftarrow H_K(Z)$;
  $\chi \leftarrow E_{K_0}(Y')$;
  send $(\textsf{response}, R, \chi)$;

Function *msg-response*(μ):
  $(R', \chi') \leftarrow \mu$;
  if $R' \notin C$ **then return**;
  $Z \leftarrow rR'$;
  $(K_0, K_1) \leftarrow H_K(Z)$;
  $Y' \leftarrow D_{K_0}(\chi')$;
  if $Y' \neq Y$ **then return**;
  output K_1; **stop**;

Figure 4.2: Formalization of W-kx

the adversary, since a group element $R' \in C$ if and only if the session S responded to some message $(\textsf{challenge}, R', c')$.

The value $R = rP$ is easily computed given r, and besides is sent in the clear in the session's first message. The expected response $Y = rX'$ is also easily computed from r. The converse is also true, since r can be recovered from R and c in the session's challenge message and the value Y. Besides, r is necessary for computing Z in response to incoming challenges.

We conclude that the only 'interesting' session state is r.

Security Having formally presented the protocol, we can now state our main theorem about its security. The proof is given in appendix A.1.

4.3.1 Theorem (SK-security of W-kx) *Let G be a cyclic group. Let $\mathcal{E} = (\kappa, E, D)$ be a symmetric encryption scheme. Then*

$$\mathbf{InSec}^{\text{sk}}(W\text{-}kx^{G,\mathcal{E}}; t, n, q_S, q_M, q_I, q_K) \leqslant 2q_S\big(\mathbf{InSec}^{\text{ind-cca}}(\mathcal{E}; t', q_M, q_M) + \mathbf{InSec}^{\text{mcdh}}(G; t', q_K) + n\,\mathbf{InSec}^{\text{mcdh}}(G; t', q_M + q_I)\big) + \frac{n(n-1)}{|G|} + \frac{2q_M}{2^{\ell_I}}.$$

where $t' = t + O(n) + O(q_S) + O(q_M q_I) + O(q_K)$.

4.4 Insecure protocol variants

It's important to feed the session-id and verifier's public key to the random oracle H_I when constructing the check-value c. Without these, the protocol is vulnerable to attack. In this section, we consider some variants of our protocol which hash less information, and show attacks against them.

To simplify the presentation, we shall consider Alice and Bob, and a third character Carol. We shall be attacking a pair of matching sessions A and B, run by Alice and Bob respectively. Let Alice and Bob's private keys be x_A and x_B, and let their public keys be $X_A = x_A P$ and $X_B = x_B P$ respectively.

Protocol diagram notation In order to keep the presentation as clear as possible, we use a simple diagrammatic notation for describing protocol runs. A line of the form

$$\textit{action} \longrightarrow S \longrightarrow \textit{result}$$

states that the adversary performs the given *action* on session S, with the stated *result*. The *action* may be

- Create session (P_i, P_j, s): the session is created, running in party P_i, with partner P_j and session-id s.
- Receive μ: the session is activated with an incoming message μ.
- Session-state reveal: The adversary requests the session's internal state.

The *result* may be

- Send μ': the session requests the delivery of the message μ'.
- Complete: K: the session completes, outputting the key K.
- State: σ: the session's state is revealed to be σ.
- (Ignore): the result of the action is unimportant.

Omitting the session-id Consider a protocol variant where session S sets $h_S = H_I(X_N, R_S, Y_S)$, where N is the session's partner. That is, we've omitted the session-id from the hash. An adversary can cross over two sessions between Alice and Bob. Here's how.

The attack assumes that Alice and Bob set up two pairs of matching sessions with each other, thus.

$$\begin{aligned}
\text{Create session } (\text{Alice}, \text{Bob}, s) &\longrightarrow A \longrightarrow \text{Send (challenge, } R_A, c_A) \\
\text{Create session } (\text{Bob}, \text{Alice}, s) &\longrightarrow B \longrightarrow \text{Send (challenge, } R_B, c_B) \\
\text{Create session } (\text{Alice}, \text{Bob}, s') &\longrightarrow A' \longrightarrow \text{Send (challenge, } R_{A'}, c_{A'}) \\
\text{Create session } (\text{Bob}, \text{Alice}, s') &\longrightarrow B' \longrightarrow \text{Send (challenge, } R_{B'}, c_{B'})
\end{aligned}$$

Observe that the session pairs use distinct session-ids $s \neq s'$, so this is allowed. Now the adversary crosses over the challenges, using the second pair of sessions to provide responses to the challenges issued by the first pair. By revealing the state in the second pair of sessions, the adversary can work out the (probably different) session keys accepted by the first pair.

$$\begin{array}{rcl}
\text{Receive (challenge, } R_B, c_B) & \longrightarrow A' \longrightarrow & \text{Send (response, } R_{A'}, E_{K_{A'B,0}}(x_A R_B)) \\
\text{Receive (challenge, } R_A, c_A) & \longrightarrow B' \longrightarrow & \text{Send (response, } R_{B'}, E_{K_{B'A,0}}(x_B R_A)) \\
\text{Receive (challenge, } R_{A'}, c_{A'}) & \longrightarrow A \longrightarrow & \text{(ignored)} \\
\text{Receive (challenge, } R_{B'}, c_{B'}) & \longrightarrow B \longrightarrow & \text{(ignored)} \\
\text{Session-state reveal} & \longrightarrow A' \longrightarrow & r_{A'} \\
\text{Session-state reveal} & \longrightarrow B' \longrightarrow & r_{B'} \\
\text{Receive (response, } R_{B'}, E_{K_{B'A,0}}(x_B R_A)) & \longrightarrow A \longrightarrow & \text{Complete: } K_{AB',1} \\
\text{Receive (response, } R_{A'}, E_{K_{A'B,0}}(x_A R_B)) & \longrightarrow B \longrightarrow & \text{Complete: } K_{BA',1}
\end{array}$$

The adversary can now compute $K_{AB'} = H_K(r_{B'} R_A)$ and $K_{B'A} = H_K(r_{A'} R_B)$. Safely in possession of both keys, the adversary can now read and impersonate freely.

Omitting the partner's public key Now consider a protocol variant where session S sets $h_S = H_I(s, R_S, Y_S)$, where s is the session-id. An adversary can use a sessions with Carol to attack a session between Alice and Bob. Here's how the sessions are set up.

$$\begin{array}{rcl}
\text{Create session } (\text{Alice}, \text{Bob}, s) & \longrightarrow A \longrightarrow & \text{Send (challenge, } R_A, c_A) \\
\text{Create session } (\text{Bob}, \text{Alice}, s) & \longrightarrow B \longrightarrow & \text{Send (challenge, } R_B, c_B) \\
\text{Create session } (\text{Alice}, \text{Carol}, s) & \longrightarrow A' \longrightarrow & \text{Send (challenge, } R_{A'}, c_{A'}) \\
\text{Create session } (\text{Bob}, \text{Carol}, s) & \longrightarrow B' \longrightarrow & \text{Send (challenge, } R_{B'}, c_{B'})
\end{array}$$

Although each of Alice and Bob have two sessions with session-id s, this is allowed, since they are with different partners. The rest of the attack in fact proceeds identically to the previous case.

4.5 Deniability

We have claimed that the Wrestlers key-exchange protocol is *deniable*. In this section, we define what we mean, explain the limits of the deniablility of the protocol as currently presented, fix the protocol with an additional pass (which can be collapsed to a single message flow by breaking the protocol's symmetry), and prove the deniability of the resulting protocol.

Our notion of deniability is taken from Di Raimondo, Gennaro and Krawczyk [RGK06], except that, as usual, we opt for a concrete security approach.

Discussion Our definition for deniability is that, for any adversary interacting with participants in the protocol, a simulator exists which can compute the same things as the adversary. In particular, since an adversary could output a transcript of the interactions between itself and the parties, it would follow that a simulator could do this too. If the simulator is effective, its output is indistinguishable from that of the real adversary, and hence no 'judge' (distinguisher) should be persuaded by evidence presented by someone who claims to have witnessed or participated in an interaction.

We work again the model described in 4.2. That is, our adversary has complete control over the ordering and delivery of all messages. The adversary is also able, at any time, to reveal the state of any session. However, deniability is obviously impossible against an adversary who can *corrupt* other parties, since simulating such an adversary's actions would necessarily require the simulator to compute private keys corresponding to the known public keys, and this is (we believe) difficult, because an efficient algorithm for doing this could easily attack our protocol, which we already proved secure. Therefore, we forbid the adversary from corrupting parties.

In order to allow the adversary to participate in the protocol, rather than merely observing it, we need to give it one or more private keys. We could modify the initialization function *init* from figure 4.2 to give the adversary a private key, but there's an easier way: we can just give the adversary a number of private keys in its auxiliary input.

Definitions Let Π be a key-exchange protocol, in the model described in section 4.2. We use the simulation framework of section 2.10. We define the initialization function I_Π to be the initialization function of Π, as before, and the corresponding world $W_\Pi(\iota, \sigma, \tau, \mu)$ is a fairly straightforward mapping of the adversary's possible actions to the simulation model:

- The invocation new-session with $\mu = (i, j, s)$ creates a new session on party P_i, with partner P_j and session-id s. The reply $\rho = (\delta, m)$ is a *decision* $\delta \in \{\mathsf{continue}, \mathsf{abort}, \mathsf{complete}\}$ and an output message $m \in \Sigma^* \cup \{\perp\}$. If $m \neq \perp$ then m is a message to be sent to the matching session (if any).
- The invocation deliver with $\mu = (i, j, s, m)$ delivers message m to the session $S = (P_i, P_j, s)$. The response ρ is as for new-session invocations.
- The invocation reveal-session-state with $\mu = (i, j, s)$ reveals to the adversary the state of the running session $S = (P_i, P_j, s)$. The response ρ is the session's state if S is indeed a running session, or $\perp$ otherwise.
- The invocation reveal-session-key with $\mu = (i, j, s)$ reveals to the adversary the session-key of the completed session $S = (P_i, P_j, s)$. The response ρ is the session key K if the session is indeed complete, or $\perp$ otherwise.

There are no invocations corresponding to the adversary actions of corrupting parties (since deniability against an corrupting adversary is impossible, as discussed earlier), or session expiry or challenging (since they are useless in this context).

We measure the deniability of a protocol Π, using a given simulator S, by the insecurity function $\mathbf{InSec}^{\mathrm{sim}}(W_\Pi, I_\Pi, S; t_D, t_A, \mathcal{Q}_D, \mathcal{Q}_A, \mathcal{R}, \mathcal{U})$ of definition 2.10.1. The interaction bounds $\mathcal{R} = (q_S, q_M)$ we place on the adversary are on the number (q_S) of new-session and (q_M) deliver invocations it makes.

We shall (informally) say that a protocol Π is deniable if there is a simulator S_Π for which the insecurity function is small for appropriate resource bounds on the adversary and distinguisher.

The current protocol As it stands, W-*kx* isn't deniable, according to our definition, for arbitrary auxiliary inputs. Let's see why.

Suppose that Bob is an informant for the secret police, and wants to convince a judge that Alice is involved in subversive activities in cyberspace. Unfortunately, Bob's job is difficult, because of the strong zero-knowledge nature of the Wrestlers identification protocol. However, Bob can work with the judge to bring him the evidence necessary to convict Alice. Here's how.

Alice's public key is A, and Bob's public key is B. The judge chooses some session-id s, and

$r \in_\$ \mathbb{N}_{<q}$. He computes $R = rP$ and $c = r \oplus H_I(B, s, R, rA)$, and gives Bob the triple (s, R, c), keeping r secret. Bob can now persuade Alice to enter into a key-exchange with him, with session-id s. He uses (R, c) as his challenge message. When Alice sends back her response (R', χ) (because Bob's challenge is correctly formed), Bob aborts and shows (R', χ) to the judge. The judge knows r and can therefore decrypt χ and check the response. If it's wrong, then Bob was cheating and gets demoted – he can't get the right answer by himself because that would require him to impersonate Alice. If it's right, Alice is really a subversive element and 'disappears' in the night.

We shall show in theorem 4.5.2 below that this is basically the only attack against the deniability of the protocol. However, we can do better.

Fixing deniability We can fix the protocol to remove even the attack discussed above. The change is simple: we feed *both* parties' challenges to the hash function H_I rather than just the sender's. We use a five-argument hash function (random oracle) $H_I : G^2 \times \Sigma^{\ell_S} \times G^2 \to \Sigma^{\ell_I}$. We introduce a new message pass at the beginning of the protocol: each session simply sends its challenge point $R = rP$ in the clear as a 'pre-challenge'. The actual challenge is R and $c = r \oplus H_I(X, R', s, R, c)$, where R' is the challenge of the matching session.

By breaking symmetry, we can reduce the communication complexity of this variant to four messages. As before, we analyse the symmetrical version. The extra flow might seem a high price to pay, but we shall see that it has additional benefits beyond deniability.

A summary of the new protocol is shown in figure 4.3, and the formal description is shown in figure 4.4.

The security of this variant is given by the following theorem, whose proof is in appendix A.2.

4.5.1 Theorem (SK-security of W-*dkx*) *Let G be a cyclic group. Let $\mathcal{E} = (\kappa, E, D)$ be a symmetric encryption scheme. Then*

$$\mathbf{InSec}^{\mathrm{sk}}(W\text{-}dkx^{G,\mathcal{E}}; t, n, q_S, q_M, q_I, q_K) = \mathbf{InSec}^{\mathrm{sk}}(W\text{-}kx^{G,\mathcal{E}}; t, n, q_S, q_M, q_I, q_K)$$

Deniability of the Wrestlers protocols In order to quantify the level of deniability our protocols provide, we shall impose a limit on the auxiliary input to the adversary. In particular, we shall use $\mathcal{U}$ of definition 2.10.1 to count the number of *challenges* in the auxiliary input. That is, $\mathcal{U} = n_C$ is the number of tuples (i, j, s, R', R, c) for which there is an r such that $R = rP$ and $c = r \oplus H_I(R', X_j, s, R, rX_i)$ (or without the R' for W-*kx*).

With this restriction in place, we can state the following theorem about the deniability of our protocols.

4.5.2 Theorem (Deniability of W-*kx* and W-*dkx*) *There exist simulators $S_{W\text{-}kx}$ and W-dkx such that*

$$\mathbf{InSec}^{\mathrm{sim}}(W_{W\text{-}kx^{G,\mathcal{E}}}, I_{W\text{-}kx^{G,\mathcal{E}}}, S_{W\text{-}kx^{G,\mathcal{E}}}; t_D, t_A, \mathcal{Q}_D, \mathcal{Q}_A, (q_S, q_M), 0) \leqslant \frac{q_M}{2^{\ell_I}}$$

and

$$\mathbf{InSec}^{\mathrm{sim}}(W_{W\text{-}dkx^{G,\mathcal{E}}}, I_{W\text{-}dkx^{G,\mathcal{E}}}, S_{W\text{-}dkx^{G,\mathcal{E}}}; t_D, t_A, \mathcal{Q}_D, \mathcal{Q}_A, (q_S, q_M), n_C) \leqslant \frac{n_C q_S}{|G|} + \frac{q_M}{2^{\ell_I}}.$$

The running time of the simulators is $O(t_A) + O(\mathcal{Q}_A q_M)$.

Proof The simulators $S_{W\text{-}kx}$ and $S_{W\text{-}dkx}$ are very similar. We describe both here. Both are fake-world simulators, working as follows.

Setup	Group $G = \langle P \rangle$; $\lvert G \rvert = q$ is prime. $H_I(\cdot, \cdot, \cdot, \cdot, \cdot)$ and $H_K(cdot)$ are secure hashes. $\mathcal{E} = (\kappa, E, D)$ is an IND-CCA2 symmetric encryption scheme.
Parties	U_i for $0 \leqslant i < n$.
Private keys	$x_i \in_\$ \mathbb{N}_{<q}$.
Public keys	$X_i = x_i P$.

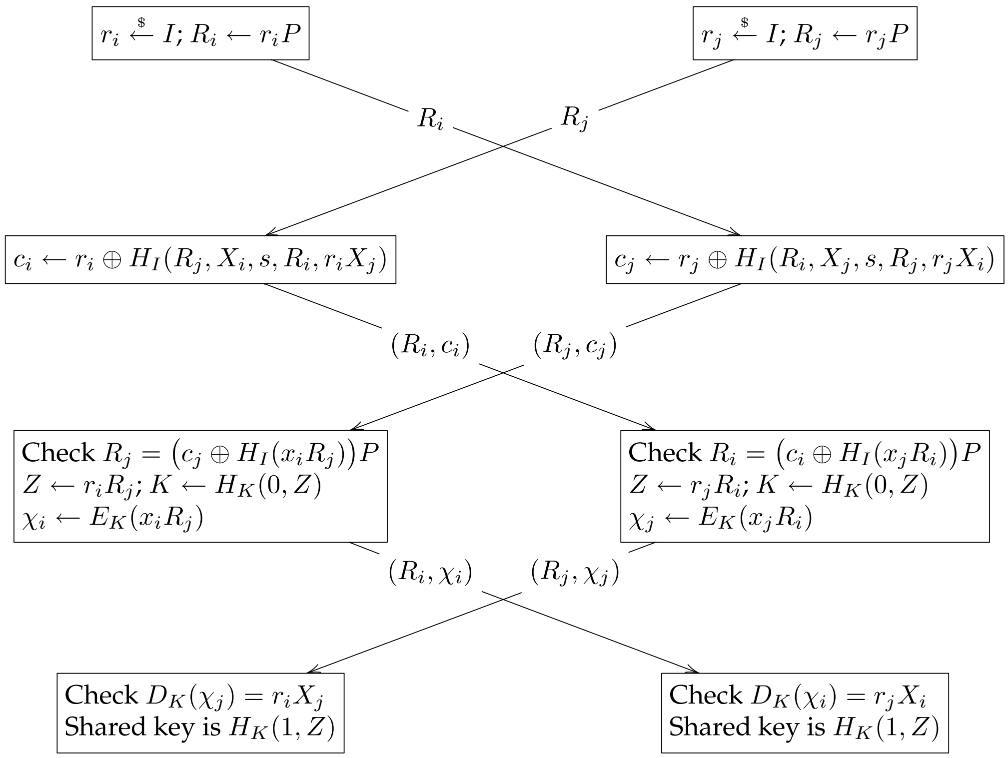

Figure 4.3: Summary of the Deniable Wrestlers Key Exchange protocol, W-dkx

1. Initially, it constructs simulated parties P_i, for $0 \leqslant i < n$, giving each the public key X_i from the common input.

2. Suppose the adversary requests creation of a new session $S = (P_i, P_j, s)$. Then the simulator creates a new session, including a random value $r_S \in_\$ \mathbb{N}_{<|G|}$, and computes $R_S = r_S P$, and $Y_S = r_S X_j$. For W-*dkx*, it sends the message $(\textsf{pre-challenge}, R_S)$; for W-*kx*, it additionally computes $h = H_I(X_i, s, R_S, Y_S)$ and sends $(\textsf{challenge}, R_S, r_S \oplus h)$.

3. Suppose, for W-*dkx*, the adversary sends a message $(\textsf{pre-challenge}, R')$ to a session $S = (P_i, P_j, s)$. The simulator computes $h = H_I(R', X_i, s, R_S, Y_S)$, and sends $(\textsf{challenge}, R_S, r_S \oplus h)$.

4. Suppose the adversary sends a message $(\textsf{challenge}, R', c')$ to session $S = (P_i, P_j, s)$. The simulator doesn't know x_i.

   (a) If $R' = R_T$ for some other simulated session T, then the simulator knows r_T such that $R_T = r_T P$. Let $Y' = r_T X_i$. The simulator computes $h = H_I(X_j, s, R', Y')$ (resp. $h = H_I(R_S, X_j, s, R', Y')$) for W-*kx* (resp. W-*dkx*) and checks that $r_T = c' \oplus h$. If not,

Function *init*(n):
  for $i \in \mathbb{N}_{<n}$ **do**
    $x \xleftarrow{\$} \mathbb{N}_{<|G|}$;
    $\mathbf{i}[i] \leftarrow x$;
    $\mathbf{p}[i] \leftarrow xP$;
  return $(\mathbf{p}, \mathbf{i})$;

Function *new-session*$^{H_I(\cdot,\cdot,\cdot,\cdot,\cdot),H_K(\cdot)}(\mathbf{p}, x, i, j, s)$:
  $X \leftarrow \mathbf{p}[i]$; $X' \leftarrow \mathbf{p}[j]$; $C \leftarrow \varnothing$;
  $r \xleftarrow{\$} \mathbb{N}_{<|G|}$; $R \leftarrow rP$; $Y \leftarrow rX'$;
  send (pre-challange, R);

Function *message*$^{H_I(\cdot,\cdot,\cdot,\cdot,\cdot),H_K(\cdot)}(\tau, \mu)$:
  if $\tau =$ pre-challenge **then** *msg-pre-challenge*(μ);
  else if $\tau =$ challenge **then** *msg-challenge*(μ);
  else if $\tau =$ response **then** *msg-response*(μ);

Function *msg-pre-challenge*(μ):
  $R' \leftarrow \mu$;
  $h \leftarrow H_I(R', X, s, R, c)$;
  $c \leftarrow r \oplus h$;
  send (*msg-challenge*, R, c);

Function *msg-challenge*(μ):
  $(R', c') \leftarrow \mu$;
  $Y' \leftarrow xR'$;
  $h' \leftarrow H_I(R, X', s, R', Y')$;
  $r' \leftarrow c' \oplus h'$;
  if $R' \neq r'P$ **then return**;
  $C \leftarrow C \cup \{R\}$;
  $Z \leftarrow rR'$;
  $(K_0, K_1) \leftarrow H_K(Z)$;
  $\chi \leftarrow E_{K_0}(Y')$;
  send (response, R, χ);

Function *msg-response*(μ):
  $(R', \chi') \leftarrow \mu$;
  if $R' \notin C$ **then return**;
  $Z \leftarrow rR'$;
  $(K_0, K_1) \leftarrow H_K(Z)$;
  $Y' \leftarrow D_{K_0}(\chi')$;
  if $Y' \neq Y$ **then return**;
  output K_1; **stop**;

Figure 4.4: Deniable key-exchange: formalization of W-dkx

the simulator discards the message. Otherwise, it computes $(K_0, K_1) = H_K(r_S R')$, and sends the message (response, $R, E_{K_0}(Y')$).

(b) Otherwise the simulator runs the extractor $T_{W\text{-}ident}$ on the adversary's history of queries $H_I(X_j, s, R', \cdot)$ (resp. $H_I(R_S, X_j, s, R', \cdot)$) for W-*kx* (resp. W-*dkx*). The extractor returns (r', Y'). If $Y' = \perp$ then the simulator ignores the message. Otherwise, the simulator computes $(K_0, K_1) = H_K(rR')$ and sends back (response, $R, E_{K_0}(Y')$).

5. Suppose the adversary sends a message (response, R', χ) to session $S = (P_i, P_j, s)$. The simulator computes $(K_0, K_1) = H_K(r_S R')$, and decrypts $Y' = D_{K_0}(\chi)$. If $Y' \neq Y_S$ then the simulator discards the message. Otherwise, it makes the simulated session complete, and outputs key K_1.

6. Finally, if the adversary reveals a session's state, the simulator reveals r_S as required; if the adversary reveals a session-key, the simulator reveals the K_1 it output.

The only point where the simulation fails to be perfect is in 4b. Let R' and c' be the values from an incoming challenge message to session $S = (P_i, P_j, s)$. Let r' be such that $R' = r'P$ and let $Y' = r'X_i$. If a random-oracle query $H_I(X_j, s, R', Y')$ (or $H_I(R_S, X_j, s, R', Y')$ for W-*dkx*) has been issued, then there are a number of possibilities. Let h' be the result of this query.

- The adversary made this query. Then the extractor will find it and return Y' if $c' = h' \oplus r'$, or $\perp$ otherwise.
- Some simulated session $U = (P_{i'}, P_{j'}, s')$ made this query. But simulated sessions only make H_I-queries when constructing challenges, so $R' = R_U$ for some session U. But the simulator does something different in that case.
- In W-*dkx*, the quadruple (s, R_S, R', c') came from the adversary's auxiliary input. In this case the simulator must fail. But $R_S = r_S P$, and r_S was chosen uniformly at random. If there are at most n_C challenge sets in the auxiliary input then this happens with probability at most $n_C/|G|$ for any given session.

We conclude that the simulator fails with probability

$$\frac{q_M}{2^{\ell_I}} + \frac{q_S n_C}{|G|}.$$

(Note that we only consider $n_C = 0$ for W-*kx*.) No adversary can distinguish the simulator from a real interaction unless the simulator fails, and the simulator is a fake-world simulator. We therefore apply proposition 2.10.4; the theorem follows. □

4.6 Practical issues

Denial of service from spoofers The adversary we considered in 4.2 is very powerful. Proving security against such a powerful adversary is good and useful. However, there are other useful security properties we might like against weaker adversaries.

Eavesdropping on the Internet is actually nontrivial. One needs control of one of the intermediate routers between two communicating parties. (There are tricks one can play involving subversion of DNS servers, but this is also nontrivial.) However, injecting packets with bogus source addresses is very easy.

Layering the protocol over TCP [Pos81a] ameliorates this problem because an adversary needs to guess or eavesdrop in order to obtain the correct sequence numbers for a spoofed packet; but the Wrestlers protocol is sufficiently simple that we'd like to be able to run it over UDP [Pos80a], for which spoofing is trivial.

Therefore, it's possible for anyone on the 'net to send Alice a spurious challenge message (R, c). She will then compute $Y = aR$, recover $r' = c \oplus H_I(\ldots, R, Y)$ check that $R = r'P$ and so on. That's at least two scalar multiplications to respond to a spoofed packet, and even with very efficient group operations, coping with this kind of simple denial-of-service attack might be difficult.

A straightforward solution is to use the Deniable variant of the protocol, and require a challenge to quote its matching session's challenge R' in its challenge. That is, upon receiving a $(\textsf{pre-challenge}, R')$, the session sends $(\textsf{challenge}, R', R, c)$. Alice then rejects any *incoming* challenge message which doesn't quote her current challenge value. Now only eavesdroppers can force her to perform expensive computations.

Indeed, one need not quote the entire challenge R': it suffices to send some short hard-to-guess hash of it, maybe just the bottom 128 bits or so.

This can't reduce security. Consider any adversary attacking this protocol variant. We can construct an adversary which attacks the original protocol just as efficiently. The new adversary attaches fake R' values to challenges output by other parties, and strips them off on delivery, discarding messages with incorrect R' values.

Key confirmation Consider an application which uses the Wrestlers protocol to re-exchange keys periodically. The application can be willing to *receive* incoming messages using the new key as soon as the key exchange completes successfully; however, it should refrain from *sending* messages under the new key until it knows that its partner has also completed. The partner may not have received the final response message, and therefore be unwilling to accept a new key; it will therefore (presumably) reject incoming messages under this new key.

While key confirmation is unnecessary for *security*, it has *practical* value, since it solves the above problem. If the application sends a switch message when it 'completes', it can signal its partner that it is indeed ready to accept messages under the new key. Our implementation sends (switch-rq, $E_{K_0}(H_S(0, R, R'))$) as its switch message; the exact contents aren't important. Our retransmission policy (below) makes use of an additional message switch-ok, which can be defined similarly.

It's not hard to show that this doesn't adversely affect the security of the protocol, since the encrypted message is computed only from public values. In the security proof, we modify the generation of response messages, so that the plaintexts are a constant string rather than the true responses, guaranteeing that the messages give no information about the actual response. To show this is unlikely to matter, we present an adversary attacking the encryption scheme by encrypting either genuine responses or fake constant strings. Since the adversary can't distinguish which is being encrypted (by the definition of IND-CCA security, definition 2.9.2), the change goes unnoticed. In order to allow incorporate our switch messages, we need only modify this adversary, to implement the modified protocol. This is certainly possible, since the messages contain (hashes of) public values. We omit the details.

However, while the extra message doesn't affect the security of our protocol, it would be annoying if an adversary could forge the switch request message, since this would be a denial of service. In the strong adversarial model, this doesn't matter, since the adversary can deny service anyway, but it's a concern against less powerful adversaries. Most IND-CCA symmetric encryption schemes also provide integrity of plaintexts [BN00] (e.g., the encrypt-then-MAC generic composition approach [BN00, Kra01], and the authenticated-encryption modes of [RBB03, BRW04, MV04]), so this isn't a great imposition.

Optimization and piggybacking We can optimize the number of messages sent by combining them. Here's one possible collection of combined messages:

pre-challenge	R
challenge	R', R, $c = H_I(R', X, s, R, c) \oplus r$
response	R', R, c, $E_{K_0}(xR')$
switch	R', $E_{K_0}(xR', H_S(0, R, R'))$
switch-ok	R', $E_{K_0}(H_S(1, R, R'))$

The combination is safe:

- the switch and switch-ok messages are safe by the argument above; and

- the other recombinations can be performed and undone in a 'black box' way, by an appropriately defined SK-security adversary.

Unreliable transports The Internet UDP [Pos80a] is a simple, unreliable protocol for transmitting datagrams. However, it's very efficient, and rather attractive as a transport for datagram-based applications such as virtual private networks (VPNs). Since UDP is a best-effort rather than a reliable transport, it can occasionally drop packets. Therefore it is necessary for a UDP application to be able to retransmit messages which appear to have been lost.

We recommend the following simple retransmission policy for running the Wrestlers protocol over UDP.

- Initially, send out the pre-challenge message every minute.
- On receipt of a pre-challenge message, send the corresponding full challenge, but don't retain any state.
- On receipt of a (valid) challenge, record the challenge value R' in a table, together with $K = (K_0, K_1)$ and the response $Y' = xR'$. If the table is full, overwrite an existing entry at random. Send the corresponding response message, and retransmit it every ten seconds or so.
- On receipt of a (valid) response, discard any other challenges, and stop sending pre-challenge and response retransmits. At this point, the basic protocol described above would *accept*, so the key K_1 is known to be good. Send the switch message, including its response to the (now known-good) sender's challenge.
- On receipt of a (valid) switch, send back a switch-ok message and stop retransmissions. It is now safe to start sending messages under K_1.
- On receipt of a (valid) switch-ok, stop retransmissions. It is now safe to start sending messages under K_1.

Key reuse Suppose our symmetric encryption scheme $\mathcal{E}$ is not only IND-CCA secure (definition 2.9.2) but also provides integrity of plaintexts [BN00] (or, alternatively, is an AEAD scheme [Rog02]. Then we can use it to construct a secure channel, by including message type and sequence number fields in the plaintexts, along with the message body. If we do this, we can actually get away with just the one key $K = H_K(Z)$ rather than both K_0 and K_1.

To do this, it is essential that the plaintext messages (or additional data) clearly distinguish between the messages sent as part of the key-exchange protocol and messages sent over the 'secure channel'. Otherwise, there is a protocol-interference attack: an adversary can replay key-exchange ciphertexts to insert the corresponding plaintexts into the channel.

We offer a sketch proof of this claim in appendix A.3.

5 Conclusions

We have presented new protocols for identification and authenticated key-exchange, and proven their security. We have shown them to be efficient and simple. We have also shown that our key-exchange protocol is deniable. Finally, we have shown how to modify the key-exchange protocol for practical use, and proven that this modification is still secure.

6 Acknowledgements

The Wrestlers Protocol is named after the Wrestlers pub in Cambridge where Clive Jones and I worked out the initial design.

7 References

[BAN89] Michael Burrows, Martin Abadi, and Roger Needham; *A logic of authentication*; Tech. Rep. 39; Digital Equipment Corporation, Systems Research Centre; February 1989.

[BCK96] M. Bellare, R. Canetti, and H. Krawczyk; *Keying hash functions for message authentication*; in Neal Koblitz, ed., *Advances in cryptology, CRYPTO '96: 16th annual international cryptology conference, Santa Barbara, California, USA, August 18–22, 1996: proceedings*; vol. 1109 of *Lecture Notes in Computer Science*; Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc.; 1996; ISBN 3-540-61512-1; ISSN 0302-9743; pp. 1–15; URL `Fullversion:http://www.research.ibm.com/security/`; sponsored by the International Association for Cryptologic Research (IACR), in cooperation with the IEEE Computer Society Technical Committee on Security and Privacy and the Computer Science Department of the University of California at Santa Barbara (UCSB).

[BCK98] Mihir Bellare, Ran Canetti, and Hugo Krawczyk; *A modular approach to the design and analysis of key exchange protocols*; in *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98)*; ACM Press, New York; May 23–26 1998; ISBN 0-89791-962-9; pp. 419–428; URL `http://www.cs.ucsd.edu/~mihir/papers/key-distribution.html`.

[BDJR97] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway; *A concrete security treatment of symmetric encryption*; in *IEEE Symposium on Foundations of Computer Science*; 1997; pp. 394–403; URL `http://www-cse.ucsd.edu/users/mihir/papers/sym-enc.html`.

[BDPR98] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway; *Relations among notions of security for public-key encryption schemes*; *Lecture Notes in Computer Science*; 1462; 1998: 26–??; ISSN 0302-9743.

[Bel99] M. Bellare; *Practice-oriented provable security*; *Lecture Notes in Computer Science*; 1561; 1999: 1–15; ISSN 0302-9743.

[BF03] Dan Boneh and Matthew Franklin; *Identity-based encryption from the Weil pairing*; *SIAM Journal on Computing*; 32 (3); June 2003: 586–615; ISSN 0097-5397 (print), 1095-7111 (electronic); URL `http://epubs.siam.org/sam-bin/dbq/article/39852`.

[BGR95] Mihir Bellare, Roch Guerin, and Phillip Rogaway; *XOR MACs: New methods for message authentication using finite pseudorandom functions*; in Don Coppersmith, ed., *Advances in cryptology, CRYPTO '95: 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27–31, 1995: proceedings*; vol. 963 of *Lecture Notes in Computer Science*; Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc.; 1995; ISBN 3-540-60221-6 (Berlin); ISSN 0302-9743; pp. 15–??; sponsored by the International Association for Cryptologic Research

(IACR), in cooperation with the IEEE Computer Society Technical Committee on Security and Privacy.

[BKR94] Mihir Bellare, Joe Kilian, and Phillip Rogaway; *The security of cipher block chaining*; in Desmedt [Des94]; pp. 341–358.

[BN00] Mihir Bellare and Chanathip Namprempre; *Authenticated encryption: relations among notions and analysis of the generic composition paradigm*; in *Advances in cryptology—ASIACRYPT 2000 (Kyoto)*; vol. 1976 of *Lecture Notes in Comput. Sci.*; Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc.; 2000; pp. 531–545.

[Bon98] D. Boneh; *The decision Diffie-Hellman problem*; *Lecture Notes in Computer Science*; 1423; 1998: 48–63; ISSN 0302-9743; URL `http://theory.stanford.edu/~dabo/papers/DDH.ps.gz`.

[BR93] Mihir Bellare and Phillip Rogaway; *Random oracles are practical*; in *Proceedings of the First Annual Conference on Computer and Communications Security*; ACM; 1993; URL `http://www-cse.ucsd.edu/users/mihir/papers/ro.html`.

[BR94] M. Bellare and P. Rogaway; *Entity authentication and key distribution*; in Desmedt [Des94]; pp. 232–249.

[BR95a] M. Bellare and P. Rogaway; *Optimal asymmetric encryption*; in Alfredo De Santis, ed., *Advances in cryptology — EUROCRYPT '94: Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9–12, 1994: proceedings*; vol. 950 of *Lecture Notes in Computer Science*; Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc.; 1995; ISBN 3-540-60176-7; ISSN 0302-9743; pp. 92–111.

[BR95b] M. Bellare and P. Rogaway; *Provably secure session key distribution: The three party case*; in ACM, ed., *Proceedings of the twenty-seventh annual ACM Symposium on Theory of Computing: Las Vegas, Nevada, May 29–June 1, 1995*; ACM Press, New York, NY 10036, USA; 1995; ISBN 0-89791-718-9; pp. 57–66; aCM order no. 508950.

[BR96] M. Bellare and P. Rogaway; *The exact security of digital signatures — how to sign with RSA and Rabin*; *Lecture Notes in Computer Science*; 1070; 1996: 399–??; ISSN 0302-9743.

[BR04] Mihir Bellare and Phillip Rogaway; *Code-based game-playing proofs and the security of triple encryption*; Cryptology ePrint Archive, Report 2004/331; 2004; URL `http://eprint.iacr.org/2004/331`; full version of [BR06].

[BR06] Mihir Bellare and Phillip Rogaway; *The security of triple encryption and a framework for code-based game-playing proofs*; in Serge Vaudenay, ed., *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*; vol. 4004 of *Lecture Notes in Computer Science*; Springer; 2006; ISBN 3-540-34546-9; pp. 409–426; proceedings version of [BR04].

[BRW04] Mihir Bellare, Phillip Rogaway, and David Wagner; *The EAX mode of operation*; in Bimal K. Roy and Willi Meier, eds., *FSE*; vol. 3017 of *Lecture Notes in Computer Science*; Springer; 2004; ISBN 3-540-22171-9; pp. 389–407; URL `http://www.cs.berkeley.edu/~daw/papers/eax-fse04.ps`.

[BWJM97] S. Blake-Wilson, D. Johnson, and A. Menezes; *Key agreement protocols and their security analysis*; *Lecture Notes in Computer Science*; 1355; 1997: 30–??; ISSN 0302-9743; URL `http://www.cacr.math.uwaterloo.ca/~ajmeneze/publications/agreement.ps`.

[BWM98] S. Blake-Wilson and A. Menezes; *Entity authentication and authenticated key transport protocols employing asymmetric techniques*; *Lecture Notes in Computer Science*; 1361; 1998: 137–??; ISSN 0302-9743; URL `http://www.cacr.math.uwaterloo.ca/~ajmeneze/publications/transport.ps`.

[Can01] Ran Canetti; *Universally composable security: A new paradigm for cryptographic protocols*; Report 2000/067; Cryptology ePrint Archive; October 2001; URL `http://eprint.iacr.org/2000/067`; extended Abstract appeared in proceedings of the 42nd Symposium on Foundations of Computer Science (FOCS), 2001.

[Cer00] Certicom Research; *Standards for efficient cryptography, SEC 1: Elliptic curve cryptography, version 1.0*; 2000; URL `http://www.secg.org/download/aid-385/sec1_final.pdf`.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi; *The random oracle methodology, revisited*; *Journal of the ACM*; 51 (4); July 2004: 557–594; ISSN 0004-5411.

[CK01] Ran Canetti and Hugo Krawczyk; *Analysis of key-exchange protocols and their use for building secure channels*; May 2001; URL `http://eprint.iacr.org/2001/040`; an extended abstract appears in the proceedings of Eurocrypt 2001.

[CK02] Ran Canetti and Hugo Krawczyk; *Universally composable notions of key exchange and secure channels*; *Lecture Notes in Computer Science*; 2332; 2002: 337–??; ISSN 0302-9743; URL `http://eprint.iacr.org/2002/059`.

[Des94] Yvo G. Desmedt, ed.; *Advances in cryptology, CRYPTO '94: 14th annual international cryptology conference, Santa Barbara, California, USA, August 21–25, 1994: proceedings*; vol. 839 of *Lecture Notes in Computer Science*; Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc.; 1994; ISBN 3-540-58333-5 (Berlin), 0-387-58333-5 (New York); ISSN 0302-9743.

[ElG85] Taher ElGamal; *A public key cryptosystem and a signature scheme based on discrete logarithms*; *Lecture Notes in Computer Science*; 196; 1985: 10–18; ISSN 0302-9743.

[KM06] Neal Koblitz and Alfred Menezes; *Another look at "provable security". ii*; Cryptology ePrint Archive, Report 2006/229; 2006; URL `http://eprint.iacr.org/2006/229`.

[Kra01] Hugo Krawczyk; *The order of encryption and authentication for protecting communications (or: how secure is SSL?)*; June 2001; URL `http://eprint.iacr.org/2001/045`; an abridged version appears in the proceedings of CRYPTO 2001.

[Men05] Alfred Menezes; *An introduction to pairing-based cryptography*; 2005; URL `http://www.cacr.math.uwaterloo.ca/~ajmeneze/publications/pairings.pdf`; notes from lectures given in Santander, Spain.

[MV04] David A. McGrew and John Viega; *The security and performance of the galois/counter mode (GCM) of operation*; in Anne Canteaut and Kapalee Viswanathan, eds., *INDOCRYPT*; vol. 3348 of *Lecture Notes in Computer Science*; Springer; 2004; ISBN 3-540-24130-2; pp. 343–355; URL `http://eprint.iacr.org/2004/193`.

[Pos80a] J. Postel; *RFC 768: User datagram protocol*; August 1980; URL ftp://ftp.internic.net/rfc/rfc768.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc768.txt; status: STANDARD. See also STD0006 [Pos80b].

[Pos80b] J. Postel; *STD 6: User Datagram Protocol*; August 1980; URL ftp://ftp.isi.edu/in-notes/rfc768.txt;ftp://ftp.isi.edu/in-notes/std/std6.txt;ftp://ftp.math.utah.edu/pub/rfc/rfc768.txt;ftp://ftp.math.utah.edu/pub/rfc/std/std6.txt; see also RFC0768 [Pos80a].

[Pos81a] J. Postel; *RFC 793: Transmission control protocol*; September 1981; URL ftp://ftp.internic.net/rfc/rfc793.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc793.txt; see also STD0007 [Pos81b]. Status: STANDARD.

[Pos81b] J. Postel; *STD 7: Transmission Control Protocol: DARPA Internet Program Protocol Specification*; September 1981; URL ftp://ftp.isi.edu/in-notes/rfc793.txt;ftp://ftp.isi.edu/in-notes/std/std7.txt;ftp://ftp.math.utah.edu/pub/rfc/rfc793.txt;ftp://ftp.math.utah.edu/pub/rfc/std/std7.txt; see also RFC0793 [Pos81a].

[RBB03] Phillip Rogaway, Mihir Bellare, and John Black; *OCB: a block-cipher mode of operation for efficient authenticated encryptiona*; *ACM Transactions on Information and System Security*; 6 (3); 2003: 365–403; URL http://www.cs.colorado.edu/~jrblack/papers/ocb.pdf.

[RGK06] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk; *Deniable authentication and key exchange*; Cryptology ePrint Archive, Report 2006/280; 2006; URL http://eprint.iacr.org/2006/280.

[Rog02] Phillip Rogaway; *Authenticated-encryption with associated-data*; in Ravi Sandhu, ed., *Proceedings of the 9th ACM Conference on Computer and Communications Security*; ACM Press, Washington, DC, USA; November 2002; pp. 98–107; URL http://www.cs.ucdavis.edu/~rogaway/papers/ad.html.

[Sch96] Bruce Schneier; *Applied Cryptography: Protocols, Algorithms, and Source Code in C*; John Wiley and Sons, Inc., New York, NY, USA; Second edn.; 1996; ISBN 0-471-12845-7 (cloth), 0-471-11709-9 (paper); URL http://www.counterpane.com/applied.html.

[Sho97] Victor Shoup; *Lower bounds for discrete logarithms and related problems*; in Walter Fumy, ed., *Advances in cryptology — EUROCRYPT '97: International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11–15, 1997: proceedings*; vol. 1233 of *Lecture Notes in Computer Science*; Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc.; 1997; ISBN 3-540-62975-0; ISSN 0302-9743; pp. xi + 507; URL http://www.shoup.net/papers/; sponsored by the International Association for Cryptologic Research (IACR).

[Sho99] Victor Shoup; *On formal models for secure key exchange*; April 21 1999; URL http://www.shoup.net/papers/skey.ps.Z.

[Sho01] Victor Shoup; *OAEP reconsidered*; *Lecture Notes in Computer Science*; 2139; 2001: 239–??; ISSN 0302-9743; URL http://link.springer-ny.com/link/service/series/0558/bibs/2139/21390239.htm;http://link.springer-ny.com/link/service/series/0558/papers/2139/21390239.pdf.

[Sho04] Victor Shoup; *Sequences of games: a tool for taming complexity in security proofs*; Cryptology ePrint Archive, Report 2004/332; 2004; URL `http://eprint.iacr.org/2004/332`.

[SW06] D.R. Stinson and J. Wu; *An efficient and secure two-flow zero-knowledge identification protocol*; Cryptology ePrint Archive, Report 2006/337; 2006; URL `http://eprint.iacr.org/2006/337`.

A Proofs

A.1 Proof of theorem 4.3.1

Before we embark on the proof proper, let us settle on some notation. Let P_i be a party. Then we write x_i for P_i's private key and $X_i = x_i P$ is P_i's public key. Let $S = (P_i, P_j, s)$ be a session. We write r_S for the random value chosen at the start of the session, and R_S, c_S etc. are the corresponding derived values in that session.

The proof uses a sequence of games. For each game $\mathbf{G}_i$, let V_i be the event that some pair of unexposed, matching sessions both complete but output different keys, and let W_i be the event that the adversary's final output equals the game's hidden bit b^*. To save on repetition, let us write

$$\Delta_{i,j} = \max(|\Pr[V_i] - \Pr[V_j]|, |\Pr[W_i] - \Pr[W_j]|).$$

Obviously,

$$\Delta_{i,j} \leqslant \sum_{i \leqslant k < j} \Delta_{k,k+1}.$$

Here's a quick overview of the games we use.

- $\mathbf{G}_0$ is the original SK-security game.
- In $\mathbf{G}_1$, we abort the game unless all parties' public keys are distinct. Since keys are generated at random, parties are unlikely to be given the same key by accident.
- In $\mathbf{G}_2$, we change the way sessions respond to challenge messages, by using the extractor to fake up answers to most challenges. Since the extractor is good, the adversary is unlikely to notice.
- In $\mathbf{G}_3$, we abort the game if the adversary ever queries $H_K(\cdot)$ on the Diffie-Hellman secret $r_S r_T P$ shared between two unexposed matching sessions. We show that this is unlikely to happen if the Diffie-Hellman problem is hard.
- In $\mathbf{G}_4$, we abort the game if any unexposed session *accepts* a response message which wasn't sent by a matching session.

Finally, we show that the adversary has no advantage in $\mathbf{G}_4$. The theorem follows.

For ease of comprehension, we postpone the detailed proofs of some of the steps until after we conclude the main proof.

Let A be a given adversary which runs in time t, creates at most q_S sessions, delivers at most q_M messages, and makes at most q_I queries to its $H_I(\cdot,\cdot,\cdot,\cdot)$ oracle and at most q_K

queries to its $H_K(\cdot)$ oracle. Let $\mathbf{G}_0$ be the original SK-security game of definition 4.2.1, played with adversary A.

Game $\mathbf{G}_1$ is the same as game $\mathbf{G}_0$ except, if the initialization function reports two parties as having the same public key (i.e., we have $X_i \neq X_j$ where $0 \leqslant i < j < n$), we stop the game immediately and without crediting the adversary with a win. This only happens when the corresponding private keys are equal, i.e., $x_i = x_j$, and since the initialization function chooses private keys uniformly at random, this happens with probability at most $\binom{n}{2}/|G|$. Since if this doesn't happen, the game is identical to $\mathbf{G}_0$, we can apply lemma 2.4.1, and see that

$$\Delta_{0,1} \leqslant \frac{1}{|G|}\binom{n}{2} = \frac{n(n-1)}{2|G|}. \tag{A.1.1}$$

In game $\mathbf{G}_1$ and onwards, we can assume that public keys for distinct parties are themselves distinct. Note that the game now takes at most $O(q_I)$ times longer to process each message delivered by the adversary. This is where the $O(q_I q_M)$ term comes from in the theorem statement.

Game $\mathbf{G}_2$ is the same as game $\mathbf{G}_1$, except that we change the way that we make parties respond to challenge messages $(\mathsf{challenge}, R, c)$. Specifically, suppose that $S = (P_i, P_j, s)$ is a session.

- Suppose $T = (P_j, P_i, s)$ is the matching session of S. The game proceeds as before if $(R, c) = (R_T, c_T)$ is the challenge issued by T.
- Otherwise, we run the extractor $T_{W\text{-}ident}$ on the adversary's history so far of oracle queries $H_I(X_i, s, R, \cdot)$ to determine a pair (r, Y). If $r = \perp$ then we discard the message. Otherwise, we add R to the list C, and return a fake response to the adversary by computing $K = H_K(rR_S)$ and handing the adversary $(\mathsf{response}, R_S, E_K(Y))$.

The following lemma shows how this affects the adversary's probabilities of winning.

A.1.1 Lemma

$$\Delta_{1,2} \leqslant \frac{q_M}{2^{\ell_I}}. \tag{A.1.2}$$

Let us say that a session $S = (P_i, P_j, s)$ is *ripe* if

- there is a matching session $T = (P_j, P_i, s)$, and
- S is unexposed.

Suppose that S is a ripe session, and that it has a matching session T: let $Z_S = Z_T = r_S r_T P$.

Game $\mathbf{G}_3$ is the same as $\mathbf{G}_2$, except that the game is immediately aborted if ever the adversary queries its random oracle $H_K(\cdot)$ at a value Z_S for any ripe session S. The following lemma shows how this affects the adversary's probabilities of winning.

A.1.2 Lemma *For some t' within the bounds given in the theorem statement we have*

$$\Delta_{2,3} \leqslant q_S \mathbf{InSec}^{\mathrm{mcdh}}(G; t', q_K). \tag{A.1.3}$$

Game $\mathbf{G}_4$ is the same as $\mathbf{G}_3$ except that the game is immediately aborted if ever the adversary sends a response message to a ripe session S which wasn't output by its matching session as a response to S's challenge, with the result that S completes.

Let's make this more precise. Let U and V be a pair of matching sessions. Let $C_U = (\mathsf{challenge}, R_U, c_U$ be the challenge message sent by U. Let M_T be the set of messages which T has sent upon delivery of C_U. Then, in $\mathbf{G}_4$, we abort the game if, for any pair S and T of matching, unexposed sessions, S has completed as a result of being sent a message $\mu \notin M_T$. We have the following lemma.

A.1.3 Lemma *For a t' within the stated bounds, we have*

$$\Delta_{3,4} \leqslant q_S\bigl(\mathbf{InSec}^{\text{ind-cca}}(\mathcal{E}; t', q_M, q_M) + n \cdot \mathbf{InSec}^{\text{mcdh}}(G; t', q_M + q_I)\bigr) \tag{A.1.4}$$

Finally, let us consider the state we're in with $\mathbf{G}_4$.

- No ripe session completes except as a result the adversary faithfully delivering messages between it and its matching session.
- The adversary never queries Z_S for any ripe session S. If we set $K_S = (K_{S,0}, K_{S,1}) = H_K(Z_S)$, then $K_{S,1}$ is the key output by S when it completes.
- If S and T are matching ripe sessions, then $K_S = K_T$, since $Z_S = r_S R_T = r_T R_S = Z_T$.
- For any ripe session S, $K_{S,1}$ is uniformly distributed in Σ^κ and independent of the adversary's view.
- If $S = (P_i, P_j, s)$ and $T = (P_j, P_i, s)$ are matching ripe sessions, then Z_S depends only r_S and r_T. Hence, once S and T complete, and erase their states, Z_S is independent of everything except the messages sent between the two sessions. In particular, Z_S is independent of the long-term secrets x_i and x_j, so if either player is later corrupted, the key $K_{S,1}$ remains independent of the adversary's view.
- Hence, the keys output by unexposed sessions are indistinguishable from freshly-generated random strings, and remain so indefinitely.

We conclude that, for any adversary A,

$$\Pr[V_4] = 0 \qquad \text{and} \qquad \Pr[W_4] = \frac{1}{2}. \tag{A.1.5}$$

Putting equations A.1.1–A.1.5 together, we find

$$\begin{aligned}\mathbf{Adv}^{\text{sk}}_{\textit{W-ident}^{G,\mathcal{E}}}(A) \leqslant 2q_S\bigl(&\mathbf{InSec}^{\text{ind-cca}}(\mathcal{E}; t', q_M, q_M) + {} \\ &\mathbf{InSec}^{\text{mcdh}}(G; t', q_K) + n\,\mathbf{InSec}^{\text{mcdh}}(G; t', q_M + q_I)\bigr) + \frac{n(n-1)}{|G|} + \frac{2q_M}{2^{\ell_I}}\end{aligned} \tag{A.1.6}$$

The theorem follows, since A was chosen arbitrarily.

Proof of lemma A.1.1 The two games $\mathbf{G}_1$ and $\mathbf{G}_2$ differ only in whether they accept or reject particular challenge messages $(\mathsf{challenge}, R, c)$.

We claim firstly that no message is *accepted* by $\mathbf{G}_2$ which would have been rejected by $\mathbf{G}_1$. To prove the claim, it is sufficient to note that the extractor's output, if not $\perp$, is always correct, and hence if $\mathbf{G}_2$ accepts a message then $\mathbf{G}_1$ would have done so too.

Since $\mathbf{G}_2$ also behaves identically when the adversary submits to S the challenge from the matching session T, we have nothing to prove in this case. Let F be the event that the adversary submits a message $(\textsf{challenge}, R, c)$ to a session S which S would have accepted in $\mathbf{G}_1$ but would be rejected by the new rule in $\mathbf{G}_2$. By lemma 2.4.1 we have $\Delta_{1,2} \leqslant \Pr[F]$. To prove the current lemma, therefore, we must show that $\Pr[F] \leqslant q_M/2^{\ell_I}$.

Rather than consider individual challenge messages, we consider *classes* of messages. We shall refer to a quadruple $\Lambda = (i, j, s, R)$ as a *class-id*, and define some useful functions:

- the class's *session* $S(\Lambda) = (P_i, P_j, s)$;
- the class's *index* $r(\Lambda)$ is $r \in I$ where $R = rP$, which is well-defined by lemma 3.2.6;
- the class's *query* $Q(\Lambda) = (X_j, s, R, x_i R)$;
- the class's *hash* $H(\Lambda) = H_I(Q(\Lambda)) = H_I(X_j, s, R, x_i R)$;
- the class's *check-value* $c(\Lambda) = H(\Lambda) \oplus r(\Lambda)$;
- the class's *check-set* $V(\Lambda)$ is the set of check-values c such that a message $(\textsf{challenge}, R, c)$ was sent to session $S = (P_i, P_j, s)$; and
- the class's *count* $\nu(\Lambda) = |V(\Lambda)|$.

Consider any class-id $\Lambda = (i, j, s, R)$. A natural question which arises is: which participants have issued Λ's query, i.e., queried H_I at $Q(\Lambda)$?

We can characterise the $H_I(\cdot, \cdot, \cdot, \cdot)$ queries of a session $U = (P_{i'}, P_{j'}, s')$ as follows:

- computing the check-value for the challenge R_U by querying $H_I(X_{i'}, s', R_U, r_U X_{j'})$, and
- checking an incoming challenge R' by querying $H_I(X_{j'}, s', R', x_{i'} R')$.

The class Λ's query $Q(\Lambda)$ is U's check-value query if

$$(j, i, s, R) = (i', j', s', R_U)$$

i.e., U is the matching session of $S(\Lambda)$, and moreover $R = R_U$ is the challenge value issued by U. For any $c \in V(\Lambda)$, if $c = c(\Lambda)$ then $(\textsf{challenge}, R, c)$ is precisely the challenge message issued by U to $S(\Lambda)$; the rules for handling this message didn't change. However, if $c \neq c(\Lambda)$ then the message would have been rejected in $\mathbf{G}_1$, and we have already shown that $\mathbf{G}_2$ continues to reject all messages rejected by $\mathbf{G}_1$.

Let us say that a class-id $\Lambda = (i, j, s, R)$ is *bad* if

1. the value R is not the challenge issued by $S(\Lambda)$'s matching session, and
2. the adversary has not issued Λ's query $Q(\Lambda)$, *but*
3. $c(\Lambda) \in V(\Lambda)$, so one of the check-values submitted to S was actually correct.

We claim that our extractor will work perfectly unless some class-id is bad. Certainly, if R was issued by the matching session, there is nothing to prove; if the adversary has issued the relevant query then the extractor will recover $r(\Lambda)$ just fine; and if $c(\Lambda) \notin V(\Lambda)$ then all messages in the class would have been rejected by $\mathbf{G}_1$ anyway.

Let $B(\Lambda)$ be the event that the class Λ is bad. We claim that

$$\Pr[B(\Lambda)] \leqslant \frac{\nu(\Lambda)}{2^{\ell_I}}.$$

The present lemma follows, since

$$\Delta_{1,2} \leqslant \Pr[F] \leqslant \sum_{\Lambda} \Pr[B(\Lambda)] \leqslant \sum_{\Lambda} \frac{\nu(\Lambda)}{2^{\ell_I}} = \frac{1}{2^{\ell_I}} \sum_{\Lambda} \nu(\Lambda) \leqslant \frac{q_M}{2^{\ell_I}}$$

as required.

Now observe that, in $\mathbf{G}_2$, sessions don't actually check incoming challenges in this way any more – instead we run the extractor. So, to prove the claim, we consider a class Λ where properties 1 and 2 above hold. The correct hash $H(\Lambda)$ is then independent of the rest of the game, so the probability that $c(\Lambda) \in V(\Lambda)$ is precisely $\nu(\Lambda)/2^{\ell_I}$ as required.

This completes the proof the lemma. □

Proof of lemma A.1.2 Let F be the event that the adversary makes a query $H_K(Z_S)$ for some ripe session S. Since $\mathbf{G}_3$ proceeds exactly as $\mathbf{G}_2$ did unless F_2 occurs, we apply lemma 2.4.1, which tells us that $\Delta_{2,3} \leqslant \Pr[F_2]$. We must therefore bound this probability.

To do this, we consider a new game $\mathbf{G}_3'$, which is the same as $\mathbf{G}_3$, except that, at the start of the game, we choose a random number $k \in_\$ \mathbb{N}_{<q_S}$. For $0 \leqslant i < q_S$, let S_i be the ith session created by the adversary. We define F' to be the event that the adversary queries $H_K(Z_{S_k})$ when S_k is ripe.

The lemma now follows from these two claims.

1 **Claim** $\Pr[F] \leqslant q_S \Pr[F']$.

To see this, for any session S, let F_S be the event that the adversary queries $H_K(Z_S)$ when S is ripe. Then

$$\Pr[F] \leqslant \sum_{0 \leqslant i < q_S} \Pr[F_{S_i}].$$

Hence,

$$\Pr[F'] = \Pr[F_{S_k}] = \sum_{0 \leqslant i < q_S} \Pr[F_{S_i}] \Pr[k = i] = \frac{1}{q_S} \sum_{0 \leqslant i < q_S} \Pr[F_{S_i}] \geqslant \frac{\Pr[F]}{q_S}$$

proving the claim.

2 **Claim** *For some $t' = t + O(n) + O(q_S q_M) + O(q_I) + O(q_K)$, we have* $\Pr[F'] \leqslant \mathbf{InSec}^{\mathrm{mcdh}}(G; t', q_K)$.

To prove this claim, we construct an adversary B which solves the MCDH problem in G. The adversary works as follows.

1. It is given a pair $(R^*, S^*) = (r^* P, s^* P)$ of group elements; its objective is to make a verification-oracle query $V(Z^*)$ where $Z^* = r^* s^* P$.
2. It sets up a simulation of the game $\mathbf{G}_3'$, by running the *init* function, and simulating all of the parties. In particular, it chooses a random $k \in \mathbb{N}_{<q_S}$.
3. It sets up accurate simulations of the random oracles $H_K(\cdot)$ and $H_I(\cdot,\cdot,\cdot,\cdot)$, which choose random outputs for new, fresh inputs. However, whenever A queries $H_K(\cdot)$ on a group element Z, B also queries $V(Z)$.
4. It runs A in its simulated game. It responds to all of A's instructions faithfully, until the kth session-creation.

5. When creating the kth session $S = S_k = (P_i, P_j, s)$, B has party P_i choose R^* as its challenge, rather than choosing r_S and setting $R_S = r_S P$. Because it simulates all the parties, B can compute $Y_S = x_j R$, which is still correct.

6. If A requests the creation of a matching session $T = (P_j, P_i, s)$ then B has party P_j choose S^* as its challenge. Again, B computes $Y_T = x_i S^*$.

7. If A ever corrupts the parties P_i or P_j, or reveals the session state of S or T then B stops the simulation abruptly and halts.

Adversary B's running time is within the bounds required of t', and B makes at most q_K queries to $V(\cdot)$; we therefore have

$$\Pr[F'] \leqslant \mathbf{Succ}_G^{\text{mcdh}}(B) \leqslant \mathbf{InSec}^{\text{mcdh}}(G; t', q_K)$$

as required. □

Proof of lemma A.1.3 Let F_4 be the event under which we abort the game $\mathbf{G}_4$. Clearly, if F doesn't occur, games $\mathbf{G}_3$ and $\mathbf{G}_4$ proceed identically, so we can apply lemma 2.4.1 to see that $\Delta_{3,4} \leqslant \Pr[F_4]$. Bounding $\Pr[F_4]$, however, is somewhat complicated. We use a further sequence of games.

Firstly, let $\mathbf{G}_5$ be like $\mathbf{G}_4$ with the exception that we choose, at random, an integer $k \in_\$ \mathbb{N}_{<q_S}$. As we did in the proof for lemma A.1.3, let S_i be the ith session created by the adversary. For each session S_i, let T_i be its matching session, if there is one. We define F_5 to be the event that

- S_k completes immediately following delivery of a message $\mu \notin M_{T_k}$, and
- S_k was ripe at this point.

For games $\mathbf{G}_i$, for $i > 5$, we define the event F_i to be the event corresponding to F_5 in $\mathbf{G}_i$. Note that if S_k *is* sent a message in M_{T_k} then S_k immediately completes.

1 **Claim** $\Pr[F_4] \leqslant \Pr[F_5]/q_S$.

This claim is proven exactly as we did for claim 1 of lemma A.1.2.

Let $\mathbf{G}_6$ be the same as $\mathbf{G}_5$ except that we change the encrypted responses of session S_k and its matching session T_k. Let $K^* = (K_0^*, K_1^*) = H_K(Z_S)$. Then, rather than sending $(\mathsf{response}, R_S, E_{K_0^*}(Y_T))$, session S sends $(\mathsf{response}, R_S, E_{K_0^*}(1^{\ell_G}))$.

2 **Claim** $|\Pr[F_6] - \Pr[F_5]| \leqslant \mathbf{InSec}^{\text{ind-cca}}(\mathcal{E}; t', q_M, q_M)$.

To prove this claim, we construct an adversary B which attacks the IND-CCA security of our encryption scheme $\mathcal{E}$. The adversary B works as follows.

1. It is given no input, but a pair of oracles $E(\cdot, \cdot)$ and $D(\cdot)$; the former encrypts either the left or right input, according to a hidden bit, and the latter decrypts ciphertexts other than those returned by $E(\cdot, \cdot)$. Its goal is to guess the hidden bit.

2. It sets up a simulation of the game $\mathbf{G}_5$, by running the *init* function, and simulating all of the parties. In particular, it chooses a random $k \in \mathbb{N}_{<q_S}$.

3. It sets up accurate simulations of the random oracles $H_K(\cdot)$ and $H_I(\cdot, \cdot, \cdot, \cdot)$.

4. It runs A in its simulated game. It responds to all of A's instructions faithfully, except for the matching sessions S_k and T_k. Let $S = S_k = (P_i, P_j, s)$, and $T = T_k = (P_j, P_i, s)$.

5. Suppose T is sent the message $C_S = (\textsf{challenge}, R_S, c_S)$. Rather than computing $K^* = H_K(r_T R_S)$ and performing the encryption properly, B queries its left-or-right encryption oracle $E(\cdot, \cdot)$ on $E(1^{\ell_G}, x_j R_S)$, and sends the resulting ciphertext χ back to S as part of a message $(\textsf{response}, R_T, \chi)$. The set M_T is precisely the set of messages constructed in this fashion. (Recall that challenge messages other than C_S aren't actually delivered to T, since we simulate the responses using the extractor, as of $\mathbf{G}_2$.)

6. Suppose S is sent a message $M = (\textsf{response}, R_T, \chi) \in M_T$. We immediately stop the simulation, and B returns 0.

7. Suppose, instead, that S is sent some message $M' = (\textsf{response}, R, \chi) \notin M_T$. There are two cases to consider. If $R = R_T$ then we must have χ distinct from the ciphertexts returned by the $E(\cdot, \cdot)$ oracle, so we can invoke the decryption oracle $D(\cdot)$ on χ to obtain a response Y. Alternatively, if $R \neq R_T$, we can compute the key $K = (K_0, K_1) = H_K(r_S R)$, and recover $Y = D_{K_0}(\chi)$. In either case, if $Y = r_S X_j)$ then S would complete at this point: B stops the simulation and returns 1.

8. If A exposes S (by corrupting P_i or P_j, or revealing S or T) then we stop the simulation and B returns 0.

9. Finally, if the game stops, either because A halts, or because of one of the special rules introduced in earlier games, B returns 0.

It is clear that B's oracle queries are acceptable, since $|x_j R_S| = \ell_G$ by definition, and B never queries $D(\cdot)$ on a ciphertext returned by its encryption oracle. By the rules of $\mathbf{G}_3$, we know that the game stops immediately if A ever queries Z_S, so the key K^* is independent of everything in A's view except the ciphertexts χ output by S and T. Therefore, if the hidden bit of the IND-CCA game is 1, B accurately simulates $\mathbf{G}_5$, whereas if the bit is 0 then B accurately simulates $\mathbf{G}_6$. We issue no more that q_M encryption or decryption queries. Finally, B's running time is within the bounds allowed for t'. Therefore,

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{E}}(B) = \Pr[F_5] - \Pr[F_6] \leqslant \mathbf{InSec}^{\text{ind-cca}}(\mathcal{E}; t', q_M, q_M).$$

We construct the adversary $\bar{B}$ which is the same as B above, except that $\bar{B}$ returns 0 whenever B returns 1, and *vice versa*. Clearly

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{E}}(\bar{B}) = (1 - \Pr[F_5]) - (1 - \Pr[F_6]) = \Pr[F_6] - \Pr[F_5] \leqslant \mathbf{InSec}^{\text{ind-cca}}(\mathcal{E}; t', q_M, q_M).$$

This proves the claim.

Let $\mathbf{G}_7$ be the same as $\mathbf{G}_6$, except that at the start of the game we choose a random $m \in \mathbb{N}_{<n}$, and when the adversary creates the session $S = S_k = (P_i, P_j, s)$, we abort the game unless $j = m$. Clearly we have $\Pr[F_6] = n \Pr[F_7]$.

Finally, we can explicitly bound F_6. In $\mathbf{G}_6$, the adversary's view is independent of the correct response $Y_S = r_S X_S = x_j R_S$ to S's challenge. Therefore, if A manages to send any message $\mu \notin M_T$ which causes S to complete, then it has impersonated P_j.

3 Claim $\Pr[F_7] \leqslant \mathbf{InSec}^{\text{mcdh}}(G; t', q_M + q_I)$.

The present lemma follows from this and the previous claims.

To prove the claim formally, we construct an adversary B', which behaves as follows.

1. It is given as input a public key X^* and a single challenge (R^*, c^*), a random oracle $H_I^*(\cdot, \cdot)$, and an oracle $V(\cdot, \cdot)$, which verifies responses (R, Y). Its goal is to invoke $V(\cdot, \cdot)$ with a correct response to the challenge.

2. It chooses a random $k \in \mathbb{N}_{<q_S}$ and $m \in \mathbb{N}_{<n}$. It sets up a simulation of the game $\mathbf{G}_7$, by running the *init* function, and simulating all of the parties, except that it gives party P_m the public key X^*. This makes no difference, since P_m doesn't actually need to give any 'honest' responses because of the change we made in $\mathbf{G}_6$.

3. It sets up accurate simulations of the random oracles $H_K(\cdot)$ and $H_I(\cdot, \cdot, \cdot, \cdot)$, with one exception – see below.

4. It runs A in its simulated game. It responds to all of A's instructions faithfully, except for the session S_k. Let $S = S_k = (P_i, P_j, s)$, and let $T = T_k = (P_j, P_i, s)$ be its matching session.

5. When session S is created, B' checks that $j = m$, and if not stops the simulation and halts. Otherwise, B' invokes its oracle $C()$ to obtain a pair (R, c). Session S sends $C_S = (\textsf{challenge}, R, c)$ as its challenge to T.

6. When A makes a query $H_I(X^*, s, R, Y)$, B answers it by querying its *own* random oracle $H_I^*(R, Y)$.

7. When S receives a message $(\textsf{response}, R, \chi)$, we compute $(K_0, K_1) = r_S R$, and $Y = D_{K_0}(\chi)$. If $Y \neq \perp$ then B' calls $V(R, Y)$.

8. If A reveals S or corrupts P_i or P_j then B' stops the simulation immediately and halts.

The running time of B' is within the bounds required of t'; it makes at most q_I random-oracle and at most q_M verification queries. Clearly B' succeeds whenever F_7 occurs. The claim follows from theorem 3.2.1. □

A.2 Proof of theorem 4.5.1

The proof is almost identical to the proof of theorem 4.3.1, in appendix A.1. Unfortunately a black-box reduction doesn't seem possible.

We use the games and notation of section A.1.

The change to the check-value calculation doesn't affect key-generation at all, so the transition to $\mathbf{G}_1$ goes through as before.

The transition from $\mathbf{G}_1$ to $\mathbf{G}_2$ – answering challenges using the extractor – needs a little care. Let $S = (P_i, P_j, s)$ be a session, and consider an incoming message $(\textsf{challenge}, R, c)$.

- If $T = (P_j, P_i, s)$ is the matching session to S, and $R = R_T$ is the public challenge value of T, and $c = r_T \oplus H_I(R_S, X_j, s, R_T, r_T X_i)$ is the check-value output by T when it received $(\textsf{pre-challenge}, R_S)$ as input, then S replies as it did in $\mathbf{G}_1$.
- If the challenge message is any other message, then we use the extractor.

As in lemma A.1.1, we examine which sessions could have queried $H_I(R_S, X_j, s, R, x_i R)$, and for the same reasons conclude that only the matching session would have done this, and only in response to the pre-challenge R_S. It follows that $\Delta_{1,2} \leqslant q_M/2^{\ell_I}$ as before.

The remaining game steps go through unchanged. In particular, we conclude that a ripe session will only complete if the adversary has transmitted messages from its matching session correctly, and the session key is independent of the adversary's view. The theorem follows.

A.3 Sketch proof of single-key protocol for secure channels

We want to show that the Wrestlers Key-Exchange protocol, followed by use of the encryption scheme $\mathcal{E}$, with the *same* key $K = K_0$, still provides secure channels.

Secure channels definition We (very briefly!) recall the [CK01] definition of a secure channels protocol. We again play a game with the adversary. At the beginning, we choose a bit $b^* \in_\$ \{0, 1\}$ at random. We allow the adversary the ability to establish *secure channels* sessions within the parties. Furthermore, for any running session $S = (P_i, P_j, s)$, we allow the adversary to request S to send a message μ through its secure channel. Finally, the adversary is allowed to choose one ripe *challenge* session, and ask for it to send of one of a *pair* of messages (μ_0, μ_1), subject to the restriction that $|\mu_0| = |\mu_1|$; the session sends message μ_{b^*}. The adversary may not expose the challenge session.

The adversary wins if (a) it can guess the bit b^*, or (b) it can cause a ripe session S (i.e., an unexposed, running session), with a matching session T to output a message other than one that it requested that T send.

Protocol definition The protocol begins with Wrestlers key-exchange. The encryption in the key-exchange protocol is performed as $E_K(\mathsf{kx}, \cdot)$; encryption for secure channels is performed as $E_K(\mathsf{sc}, i, o, \cdot)$, where i is a sequence number to prevent replays and $o \in \{S, T\}$ identifies the sender.

Proof sketch We use the games and notation of appendix A.1.

The proof goes through as far as the step between $\mathbf{G}_5$ and $\mathbf{G}_6$ in the proof of lemma A.1.3. Here we make the obvious adjustments to our adversary against the IND-CCA security of $\mathcal{E}$. (Our adversary will need to use the left-or-right oracle for messages sent using the secure channel built on K^*. That's OK.)

In $\mathbf{G}_4$, we know that ripe sessions agree the correct key, and the adversary never queries the random oracle, so the key is independent of the adversary's view.

We define a new game $\mathbf{G}_8$, which is like $\mathbf{G}_4$, except that we stop the game if the adversary ever forges a message sent over the secure channel. That is, if a ripe session S ever announces receipt of a message not sent (at the adversary's request) by its matching session T. Let F_8 be the event that a forgery occurs. We apply lemma 2.4.1, which tells us that $\Delta_{4,8} \leqslant \Pr[F_8]$. To bound F_8, we isolate a session at random (as in lemmata A.1.2 and A.1.3), which tells us that

$$\Delta_{4,8} \leqslant q_S \cdot \mathbf{InSec}^{\text{int-ptxt}}(\mathcal{E}; t', q_M, q_M) \tag{A.3.1}$$

Finally, we can bound the adversary's advantage at guessing the hidden bit b^*. We isolate (we hope) the challenge session S by choosing a target session at random, as before. Let $K_* = H_K(Z_S)$ be the key agreed by the session (if it becomes ripe). We define an adversary B against the IND-CCA security of $\mathcal{E}$. The adversary B simulates the game. If the adversary exposes the target session, or doesn't choose it as the challenge session, B fails (and exits 0);

otherwise it uses the left-or-right encryption oracle to encrypt both of the adversary's message choices, and outputs the adversary's choice. Let b be the adversary's output, and let ε be the advantage of our IND-CCA distinguisher. Then

$$\begin{aligned}
\Pr[b = b^*] &= \Pr[b = b^* \wedge b^* = 1] + \Pr[b = b^* \wedge b^* = 0] \\
&= \frac{1}{2}\bigl(\Pr[b = b^* \mid b^* = 1] + \Pr[b = b^* \mid b^* = 0]\bigr) \\
&= \frac{1}{2}\bigl(\Pr[b = b^* \mid b^* = 1] + (1 - \Pr[b \neq b^* \mid b^* = 0])\bigr) \\
&= \frac{1}{2}\bigl(\Pr[b = 1 \mid b^* = 1] - \Pr[b = 1 \mid b^* = 0] + 1\bigr) \\
&= \frac{1}{2}\bigl(1 + q_S\,\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{E}}(B)\bigr) \\
&\leqslant \frac{1}{2}\bigl(1 + q_S\,\mathbf{InSec}^{\text{ind-cca}}(\mathcal{E}; t', q_M, q_M)\bigr).
\end{aligned} \tag{A.3.2}$$