The Diffie-Hellman Key-Agreement Scheme in the Strand-Space Model*

Jonathan C. Herzog
The MITRE Corporation
jherzog@mitre.org

Abstract

The Diffie-Hellman key exchange scheme is a standard component of cryptographic protocols. In this paper, we propose a way in which protocols that use this computational primitive can be verified using formal methods. In particular, we separate the computational aspects of such an analysis from the formal aspects. First, we use Strand Space terminology to define a security condition that summarizes the security guarantees of Diffie-Hellman. Once this property is assumed, the analysis of a protocol is a purely formal enterprise. (We demonstrate the applicability and usefulness of this property by analyzing a sample protocol.) Furthermore, we show that this property is sound in the computational setting by mapping formal attacks to computational algorithms. We demonstrate that if there exists a formal attack that violates the formal security condition, then it maps to a computational algorithm that solves the Diffie-Hellman problem. Hence, if the Diffie-Hellman problem is hard, the security condition holds globally.

1 Introduction

Consider this simplified version of the TLS [5] protocol:

1. $C \longrightarrow S : C$
2. $S \longrightarrow C : S\ [g^x]_{K_S}$
3. $C \longrightarrow S : [g^y]_{K_C}\ \{|T_1\, C\, S|\}_{K'}$
4. $S \longrightarrow C : \{|T_2\, C\, S|\}_{K'}$

where

- T_1, T_2 are fixed tags to distinguish the third message from the fourth,
- $[M]_{K_X}$ is the message M together with a signature that can be verified using the public key K_X,
- $\{|M|\}_{K'}$ is the message M encrypted with the symmetric key K',
- g is a generator for some large group G,
- x, y are randomly chosen elements of $\{1, 2, \ldots |G|\}$, and
- K' is a symmetric key created by hashing the value g^{xy}.

Informally, we note that the security of this protocol must depend on the secrecy of g^{xy} and recall the widely-known *Diffie-Hellman problem*:

> Given a group G, a generator g, g^x and g^y for x,y picked randomly from $\{1, 2, \ldots |G|\}$, calculate the value g^{xy}.

Although the complexity of the Diffie-Hellman problem is not known, there exist groups over which it is widely believed to be unsolvable (even on average) by any efficient algorithm.

However, the hardness of the Diffie-Hellman problem does not guarantee the security of this protocol. What is required is a proof, made using the assumptions and proof techniques of some model. One such model would be that of *computational cryptography*: the study of cryptography using the tools of complexity theory. A proof in this model would begin by assuming that there exists an adversary (i.e., an efficient algorithm) that can break the security of the protocol. It would then show that if such an adversary exists, there must also exist a second adversary[1] that can either forge a signature, break symmetric encryption, or solve the Diffie-Hellman problem.

Since the formal definition of the Diffie-Hellman problem (given in Section 2) is complexity-theoretic in nature, this model might be the most natural one to apply. Unfortunately, natural models are not necessarily the easiest to use. Although the computational model is sound and proofs in that model are strong, it is difficult to work in. A simpler and more intuitive framework is the *Dolev-Yao* model

*This work supported by the National Security Agency. Appears in *Proceedings, 16th IEEE Computer Security Foundations Workshop*, IEEE CS Press, June 2003.

[1] Typically using the first adversary as a component.

[6], which grew out of the formal methods community. Instead of considering all possible adversaries (as in the previous case) this model typically considers only a restricted class. In particular, the adversaries of this model operate by choosing — non-deterministically and repeatedly — from a small and explicitly enumerated set of operations. A proof of security in this model is generally a demonstration that all combinations of those operations (together with the operations performed by the honest participants) are 'safe'.

We would prefer to use the Dolev-Yao model to perform analyses. It is simple to use, and can be automated. Even when used manually, powerful general theorems allow individual protocols to be proven secure in a quick and straightforward way. However, since the Diffie-Hellman problem is computational in nature, it is not yet clear how to incorporate it into a formal approach.

In this paper we will propose one such incorporation. In particular, we attempt to separate the formal aspects of a protocol analysis from the computational ones. We do this in two steps:

1. We propose a security property which reflects (in the formal setting) the difficulty of the Diffie-Hellman problem. That is, we propose a condition which states (informally) that if honest participants use a shared secret such as g^{xy} only in certain ways, the adversary can never learn it. This property is natural and simple. It applies to a large class of real-world protocols, and is extremely useful in their analysis. (We demonstrate its use on the protocol that begins this paper.) Thus, once this property is assumed, the analysis of the protocol is firmly in the domain of formal methods.

2. However, we also show that this property can be justified using the techniques of the computational model. To this end, we give a mapping from formal attacks to computational algorithms. We show that any attack which violates the global property results in an efficient algorithm that solves the Diffie-Hellman problem. Hence, if Diffie-Hellman is hard, then no formal attack violates the property, and thus the property holds globally.

The paper is structured as follows. We begin by discussing some background material on the Diffie-Hellman problem (Section 2). We extend the Dolev-Yao model, as represented by the Strand Space method [12], to include language appropriate to the Diffie-Hellman problem (Section 3). We then introduce our security condition via an informal discussion (Section 4). We demonstrate the applicability of this security condition by using it to analyze the protocol that begins this paper (Section 5). We then justify the property by giving a mapping from formal attacks to computational algorithms (Section 6.1), and showing that any formal attack that violates the property results in a computational algorithm that solves Diffie-Hellman efficiently (Section 6.2). We finish with a discussion of related results and possible future work (Section 7).

2 The Diffie-Hellman Problem

In the simplest form of the Diffie-Hellman scheme, everyone is assumed to know a large cyclic group G and a generator g. If two entities A and B wish to agree on a secret random value, then

- A chooses a random element $x \in \{1 \ldots |G|\}$ and sends to B the value g^x, and
- B chooses a random element $y \in \{1 \ldots |G|\}$ and sends to A the value g^y.

The random value upon which they have agreed is g^{xy}, which both can calculate:

- A can calculate g^{xy} from x and g^y via $(g^y)^x = g^{xy}$ and
- B can calculate g^{xy} from y and g^x via $(g^x)^y = g^{xy}$.

Note that the scheme provides no authentication. Although A can be sure that the secret value g^{xy} is known only to A herself and the entity that generated y, A cannot tell who that entity is. Authentication and identification must be ensured by some other mechanism.

The scheme is, however, assumed to provide *secrecy* in the sense that no agent other than A and B is able to learn the value g^{xy}. This the Diffie-Hellman problem, given informally in the introduction. The *Diffie-Hellman assumption* is that the Diffie-Hellman problem is intractable for certain families of groups. More formally:[2] A *group family* is a set of finite cyclic groups $\mathcal{G} = \{G_p\}$ where p ranges over an infinite index set. The parameter p encodes the group parameters. We assume that there exists an efficient (polynomial-time) algorithm that, given p and two elements of G_p, outputs the sum of the elements. An *instance generator* for $\mathcal{G} = \{G_p\}$ is a randomized algorithm $\mathtt{IG}$ which, when given a natural number η (represented in unary), runs in time polynomial in η and outputs $\langle p, g\rangle$ where p is a random index and g is a generator for G_p. The parameter η is known as the *secruity parameter*. It is assumed that if G_p is a group generated by $\mathtt{IG}(1^\eta)$, then every element of G_p can be represented with a number of bits polynomial in η. Note that for a given η, $\mathtt{IG}(1^\eta)$ induces a distribution on the set of indices.

[2] Most of this exposition is taken from [4].

Definition 1 *The* computational[3] Diffie-Hellman assumption *is that no adversary can maintain a polynomial chance of producing g^{xy} from randomly chosen g^x and g^y as η increases:*

$\exists \mathcal{G}.\ \forall\ PPT$ *algorithms* $\mathtt{A}.\ \forall c > 0.\ \forall$ *sufficiently large* η

$$\Pr[\ \begin{array}{l}\langle p, g\rangle \leftarrow \mathtt{IG}(1^\eta);\\ x, y \leftarrow \{1, 2, \ldots, |G_p|\};\\ q^z \leftarrow \mathtt{A}(p, g, g^x, g^y):\\ g^z = g^{xy}\end{array} \ ] \leq \frac{1}{\eta^c}$$

(The notation "$\forall$ sufficiently large η" is equivalent to $\exists \eta_0 . \forall \eta \geq \eta_0$. The notation $\Pr[A; B : C]$ is equivalent to $\Pr[C|A, B]$ where A and B are experiments run in sequence. The notation $x \leftarrow D$ means the x is drawn from the distribution D. If D is a probabilistic algorithm,[4] then x is drawn from the distribution created by running that algorithm with random 'coin flips'. If D is a set, then the uniform distribution is assumed.)

The Diffie-Hellman problem and its applications have been well-studied in the world of computational cryptography. It has even gained acceptance in applied cryptography, and is used for key-agreement in such widespread protocols as SSH [13] and TLS [5]. Note, however, that the Diffie-Hellman assumption is a statement about the asymptotic nature of probabilities, and hence is inherently computational in nature. The main purpose of this work is to show how such computational statements and assumptions can be incorporated into the formal setting.

3 Strand Spaces

Rather than consider all formal protocol analysis methods, we will focus upon the Strand Space method. The standard model is described in [11, 12]; here, we focus on the extensions necessary to examine the Diffie-Hellman scheme. The extensions fall into two broad categories: extensions to the algebra and extensions to the adversary.

3.1 Extensions to the Algebra

We will extend and modify the algebra of previous strand space work in three ways.

First, we add an additional type $\mathcal{D}$ for Diffie-Hellman exchanges. We will use d_1, $d_2 \ldots$ as elements of $\mathcal{D}$, and we assume there exists an operation

$$\mathsf{DH} : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$$

to represent the Diffie-Hellman operation. We denote the range of DH by $\mathcal{D}_{DH}$. In the literature and in practice, the notation g^x may be used as both a computational and formal variable, and g^{xy} used instead of the admittedly cumbersome $\mathsf{DH}(d_1, d_2)$. In contexts where the model is clear, this overloading of notation presents no difficulty. In this work, however, we are interested in the exact relationship between the formal and computational models. Hence, we distinguish between the two by using the notation d_1, d_2 and $\mathsf{DH}(d_1, d_2)$ for the formal model, and the notation g^x, g^y and g^{xy} for the computational model.

Also, one of our ultimate goals is to enable the analysis of protocols like TLS, SSH, and the one at the beginning of this paper. These protocols use signatures and hashing, requiring us to add these two operations to the Strand Space algebra.[5] Signing a term is not assumed to hide the message in any fashion. Hashing a term is assumed to result in a key appropriate for symmetric encryption. We will assume that keys for symmetric encryption, signature generation, and signature verification are mutually disjoint, and that symmetric encryption keys created through hashing are disjoint from those created directly.

Lastly, we will later require that each element of the Strand Space algebra have a unique encoding into bit-strings. Many encryption and signature schemes, however, are inherently probabilistic. An encryption might have many different bit-string representations, and the one chosen depends on the random bits used in the encryption process. To represent this, we will add an additional type of atomic term called a *randomness* ($\mathcal{R}$). The formal encryption operator will continue to be injective, but now take randomness as an additional operator. We will also allow randomness to be in the plaintext of encryptions, filling the role of nonces in protocols.[6]

To combine and formalize these modifications:

Definition 2 *The set of terms $\mathcal{A}$ is (now) assumed to be freely generated from four disjoint sets:*

- $\mathcal{T} \subseteq \mathcal{A}$*, which contains predictable texts,*
- $\mathcal{R} \subseteq \mathcal{A}$*, which contains unpredictable random values,*
- $\mathcal{K} \subseteq \mathcal{A}$*, which contains keys, and*
- $\mathcal{D} \subseteq \mathcal{A}$*, which contains Diffie-Hellman values.*

The set of keys ($\mathcal{K}$) is divided into three disjoint sets:

- *signature keys ($\mathcal{K}_{Sig}$),*
- *verification keys ($\mathcal{K}_{Ver}$), and*

[3] As opposed to the *decisional* Diffie-Hellman assumption, which is the much stronger assumption that it is hard to distinguish the Diffie-Hellman value g^{xy} from a random group element g^z. For the rest of this paper, the "Diffie-Hellman problem" refers to the computational version.

[4] Meaning one that has access to a tape of random bits.

[5] We will not consider asymmetric encryption in this work, though we hope that it is clear how to incorporate it.

[6] Previously, we defined nonces to be a particular sub-type of texts ($\mathcal{T}$). Now, due to their special use in encryptions, we will distinguish nonces and texts.

- *keys for symmetric encryption ($\mathcal{K}_{Sym}$).*

Compound terms are built by five operations:

- hash : $\mathcal{A} \rightarrow \mathcal{K}_{Sym}$, *representing hashing into keys. We will denote the range of* hash *by* $\mathcal{K}_{hash}$.
- encr : $\mathcal{K}_{Sym} \times \mathcal{A} \times \mathcal{R} \rightarrow \mathcal{A}$, *which represents encryption. We denote the range of* encr *by* $\mathcal{E}$.
- sig : $\mathcal{K}_{Sig} \times \mathcal{A} \times \mathcal{R} \rightarrow \mathcal{A}$, *which represents signing a message. We denote the range of* sig *by* $\mathcal{S}$.
- join : $\mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$, *which represents concatenation of terms.*
- DH : $\mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$, *which represents the Diffie-Hellman operation. (As mentioned previously, We denote the range of* DH *by* $\mathcal{D}_{DH}$*.)*

We will now write $\mathsf{encr}(K, M, r)$ as $\{\!|M|\!\}^r_K$. We will also write $\mathsf{sig}(K, M, r)$ as $[M]^r_K$.

To use the machinery of the Strand Space model, we must define the subterm relation. The subterm relation in previous Strand Space work, denoted $\sqsubset$, denoted what could be learned from a message. That is, $M \sqsubset N$ iff M could be derived from N through repeated separations and decryptions:

Definition 3 *We say that M is a* subterm *of N, written $M \sqsubset N$, if:*

- $M = N$, *or*
- *if $N = N'\, N''$, then $M \sqsubset N'$ or $M \sqsubset N''$,*
- *if $N = \{\!|N'|\!\}^r_K$, then $M \sqsubset N'$,*
- *if $N = [N']^r_K$, then $M \sqsubset N'$.*

In particular, it is assumed that symmetric encryption would not reveal the key used to encrypt, so $K \not\sqsubset \{\!|M|\!\}^r_K$ unless $K \sqsubset M$.

In this work, we use a new operation, $\preceq$, to mean not only what could be learned from a term but also what must be used in its creation. To distinguish this new relation from the standard subterm relation, we use a different name:

Definition 4 *We say that M is an* ingredient *of N, written $M \preceq N$, if :*

- $M = N$, *or*
- *if $N = \mathsf{hash}\,(N') \in \mathcal{K}_{hash}$, then $M \preceq N'$,*
- *if $N = \mathsf{DH}(d_1, d_2)$, then $M \preceq d_1$ or $M \preceq d_2$,*
- *if $N = \{\!|N'|\!\}^r_K$, then $M \preceq N'$ or $M \preceq K$,*
- *if $N = [N']^r_K$, then $M \preceq N'$ or $M \preceq K$.*

Note that it is not necessarily the case that $r \preceq \{\!|M|\!\}^r_K$. While we assume that no one can produce an encryption without knowing the plaintext and key, we cannot make this same assumption about the randomness used in the encryption process.[7]

Similarly to origination, we can define the first time that a value is used on a strand:

Definition 5 *A term t* arises *on a node n iff n is an entry point to the set $I = \{t' : t \preceq t'\}$.*

3.2 Extending the Adversary

The next step in the extension of Strand Spaces is to give additional powers to the adversary. The usual way this is done in the formal methods approach is to give to the adversary some small number of unavoidable operations and to assume that the underlying cryptography ensures that no other operations are available. This is, in fact, the approach we will take with regard to the new signature and hashing operations. Regardless of the actual algorithms, the adversary can always:

- Make any predictable text,
- Make fresh random values, which we represent by allowing it to produce whatever it wants from a distinguished set $\mathcal{R}_{Adv} \subseteq \mathcal{R}$,
- Sign any value it knows with any signature key it knows,
- Extract the "plaintext" from a signed message, and
- Hash any values it knows.

Similarly, there are operations that the adversary will always be able to apply to Diffie-Hellman values:

- The adversary can always generate new Diffie-Hellman values. Hence, we distinguish the set $\mathcal{D}_P \subseteq \mathcal{D}$ and allow the adversary to generate any value in that set. (We assume that $\mathcal{D}_P$ and $\mathcal{D}_{DH}$ are disjoint.)
- Also, the adversary can always be able to perform the group operation efficiently, and hence can perform exponentiation.

Should we assume that these are the only operations available to the adversary? The answer to this question depends on whether one wishes to prove security or find flaws. If one wishes to find flaws, it makes sense to assume a limited adversary (albeit one that might have more powers than listed above). Any flaw available to such an adversary will remain

[7] Indeed, in a deterministic encryption scheme the ciphertext is completely independent of the random "input."

available to the unlimited one, and limiting the adversary's powers can make it easier to automate a flaw-finding technique. (This is the approach taken, for example, in [10].)

In this paper, however, we wish to focus on proofs of security. Hence, we wish to avoid any assumptions regarding the powers of the adversary that we cannot justify. For this reason, we will give the adversary the power to perform any efficient calculation. In keeping with the Dolev-Yao model, we assume that the underlying encryption scheme is strong enough to enforce the limited ability of the adversary to produce and manipulate ciphertexts and hashes. (Recent work [2, 1, 8, 9] has begun to justify these assumptions in terms of computational complexity.) However, the only assumption we can make about the underlying Diffie-Hellman group is that the Diffie-Hellman problem is hard. Hence, the adversary has the power to make general computations that result in Diffie-Hellman values, so long as those computations are *efficient*:

Definition 6 *A function f is* computable in probabilistic polynomial time *if there exists a probabilistic turing machine M so that for some polynomial q, for all input x, $M(x)$ terminates in time polynomial in $|x|$ and* $\Pr[M(x) = f(x)] \geq \frac{1}{q(|x|)}$.

We give to the adversary a strand for every probabilistic polynomial time-computable function from messages to group elements. The efficiency of a given function will depend upon the exact mapping from messages to bit-strings; we assume that the encoding has been fixed and that the adversary has access to every function that remains probabilistic polynomial time-calculable.

Definition 7 *A* adversary strand *is one of the following:*

M. *Text message:* $\langle +t \rangle$ *where* $t \in \mathcal{T}$

R. *Fresh randomness:* $\langle +r \rangle$ *where* $r \in \mathcal{R}_{Adv}$

C. *Concatenation:* $\langle -g, -h, +g\,h \rangle$

S. *Separation into components:* $\langle -g\,h, +g, +h \rangle$

K. *Key:* $\langle +K \rangle$ *where* $K \in \mathcal{K}_{\mathcal{P}}$.

E. *Encryption:* $\langle -K, -h, -r, +\{|h|\}^r_K \rangle$, *where* $K \in \mathcal{K}_{Sym}$

D. *Decryption:* $\langle -K, -\{|h|\}^r_K, +h \rangle$, *where* $K \in \mathcal{K}_{Sym}$

σ**.** *Signing:* $\langle -K, -h, -r + [h]^r_K \rangle$, *where* $K \in \mathcal{K}_{Sig}$

X. *Extraction of plaintext from signatures:* $\langle -[h]^r_K, +h \rangle$.

H. *Hashing:* $\langle -g, +\mathsf{hash}(g) \rangle$

F. *Fresh Diffie-Hellman value:* $\langle +d \rangle$ *where* $d \in \mathcal{D}_P$

f. *Computation of a function f:* $\langle -M_1, -M_2, -M_3, \ldots, -M_n, +d \rangle$, *where* $d_n = f(M_1, M_2, \ldots M_n)$ *and f is computable in probabilistic polynomial time.*

Although the other strands represent efficient operations, they are not subsumed by the f-strands. The f-strands only produce Diffie-Hellman values, while the other strands (except the **F** strand) produce terms of other types.

4 Derivation of the Security Property

With the preliminaries out of the way, we are now prepared to incorporate the Diffie-Hellman assumption into Strand Spaces. In particular, we will define a global condition over all bundles which represents the security guarantees provided by the Diffie-Hellman assumption. In this section, we present an informal derivation of this condition. (A more formal consideration can be found in Section 6.)

We derive our global property via the following steps:

- We propose a formal adversary "goal" which represents the act of solving the Diffie-Hellman problem.
- We argue that any attack the formal adversary can launch that accomplishes the goal without the help of honest participants will translate to an algorithm that solves the Diffie-Hellman problem. As a result, if the Diffie-Hellman problem is hard, then the formal adversary can never accomplish this goal on its own.
- We then consider the assistance that honest participants might give to the formal adversary. We provide a pair of simple and natural syntactic restrictions on honest participants, and show they ensure that the honest participants do not help the adversary achieve its goal.

We finish with the global property: if all honest participants obey the two restrictions, then the adversary can never achieve the goal.

In particular, the "goal" of our informal discussion will be

> Form a bundle where d_1, d_2 arise only on regular strands and $\mathsf{DH}(d_1, d_2)$ arises on a adversary node.

Intuitively, this goal corresponds to the situation where x and y are chosen by honest participants and kept secret. They communicate only g^x and g^y, and the adversary is somehow able to calculate g^{xy}.

Suppose that there is a bundle where the adversary is able to accomplish this goal without the help of honest participants. Without loss of generality, assume that the bundle accomplishes this goal with only two regular strands: $\langle +d_1 \rangle$

and $\langle +d_2 \rangle$. All other strands in the bundle are adversary strands. Each adversary strand represents one calculation, and each such calculation can be performed efficiently. The terms d_1, d_2, and $\mathsf{DH}(d_1, d_2)$ represent the distributions of g^x, g^y, and g^{xy} respectively (for randomly chosen x and y). Thus, the bundle represents an algorithm that takes in g^x and g^y at the two regular nodes and outputs g^{xy} at the node containing $\mathsf{DH}(d_1, d_2)$. By composing these calculations represented by the strands in the order given by the structure of the bundle, one forms an algorithm that solves the Diffie-Hellman problem. Hence, if the Diffie-Hellman problem is hard, then there exists no bundle that achieves the above goal without the use of regular strands.

What about bundles that *do* use regular strands? These may prove to be problematic. There is no restriction on the form of regular strands, after all, and so there is no prohibition against the strand (for example):

$$\langle -d_1, -d_2, f(\mathsf{DH}(d_1, d_2)) \rangle$$

where f is some easily invertible permutation on the underlying group. Although $\mathsf{DH}(d_1, d_2)$ does not originate on this regular strand, the strand allows it to originate on a adversary strand (namely, the strand that represents the computation of f^{-1}).

If regular strands can be of arbitrary form then it is possible for them to release secrets in several ways. More importantly, they can represent the computation of intractable functions. There is no reason to assume that the Diffie-Hellman problem remains hard if the adversary has access to secrets or oracles that perform inefficient calculations. Hence, it may be possible for the adversary to achieve the goal of solving Diffie-Hellman if it is assisted by (unrestricted) honest participants.

To surmount this difficulty, it is necessary to restrict our attention to those regular strands that do not provide assistance to the adversary. Intuitively, an honest participant is *simulatable*[8] [7] if any message the adversary receives from that participant is indistinguishable from one that the adversary could generate itself.[9] Hence, a simulatable honest participant is one that gives no assistance to the adversary: any help it could give would be already available.

Thus, what is required is a natural and useful class of simulatable regular strands. There are many classes from which to choose; we choose ours based on such protocols as TLS [5], SSH [13], and the one at the beginning of this paper. These protocols share two natural conditions:

1. Regular participants never calculate g^{xy} unless they know either x or y, and
2. Regular participants never actually say g^{xy}, but only use it as a source of key material.

More formally:

Definition 8 *A protocol is* conservative *with regard to Diffie-Hellman if, whenever a term* $\mathsf{DH}(d_a, d_b)$ *arises on a regular node, either* d_a *or* d_b *arises only on regular nodes.*

It would be possible to insist on a stronger connection between strands on which g^{xy} arise and the strands on which g^x and g^y arise, but it is not necessary for our purposes.

Also:

Definition 9 *We say that a protocol is* silent *with respect to Diffie-Hellman if no element of* $\mathcal{D}_{DH}$ *originates on a regular node.*

Here, we do mean "originate" and not "arise". The definition allows elements of $\mathcal{D}_{DH}$ to arise on regular nodes so long as they do not originate there. That is, a protocol is silent with respect to Diffie-Hellman if, whenever g^{xy} arises, is it as an ingredient of a symmetric key.

Both of these properties are purely syntactic, and easy to verify. Together, they ensure simulatability (as we will show in Section 6.2.) Thus, as long as regular strands meet these two properties, the adversary has no noticeable chance of achieving the original goal. We formalize this in a global property:

Definition 10 (Security Property $\mathcal{DH}$) *Suppose that* $\mathcal{B}$ *is a bundle over a protocol both silent and conservative with respect to Diffie-Hellman. If* $d_a, d_b \in \mathcal{D}$ *arise only on regular strands in* $\mathcal{B}$*, then* $\mathsf{DH}(d_a, d_b)$ *never originates in* $\mathcal{B}$*.*

We demonstrate the utility of this condition by analyzing the protocol that began this paper.

5 An Example Analysis

First, we re-visit some useful definitions and results from previous Strand Space papers, and update them for the extended algebra and adversary of Definitions 2 and 7.

Definition 11 *A set* $I \subseteq \mathcal{A}$ *is* honest *if all adversary entry points to* I *are on* ***M****,* ***R****,* ***K****,* ***F****, or* f *strands.*

Definition 12 *Let* $k \subseteq \mathcal{K}_{Sym}$*. Then a* k*-ideal of* $\mathcal{A}$ *is a set* $I \subseteq \mathcal{A}$ *such that for all* $h \in I$*,* $g \in \mathcal{A}$*,* $K \in k$*,* $r \in \mathcal{R}$*, and* $K_s \in \mathcal{K}_{Sig}$*:*

- $g\, h \in I$ *and* $h\, g \in I$*,*
- $\{\!|h|\!\}^r_K \in I$*, and*

[8] "Simulatability" here means something different than that intended by "bisimulation".

[9] Two probability distributions D_1 and D_s are indistinguishable if (informally) the output distribution of an efficient (probabilistic polynomial-time) algorithm A does not noticeably depend on whether the input was drawn from D_1 or D_2. Thus, an honest strand is simulatable if the adversary can produce a distribution indistinguishable from that of the honest participant's output, but does so without any of the honest participant's secrets or internal state.

- $[h]^r_{K_s} \in I$.

We will denote the smallest k-ideal that contains S as $I_k[S]$.

Theorem 1 *Suppose that $S \subseteq \mathcal{A}$ and $k \subseteq \mathcal{K}_{Sym}$ are such that*

- $(\mathcal{K}_{Sym} \cup \mathcal{K}_{hash}) \subseteq S \cup k$,
- $S \cap \mathcal{E} = \emptyset$,
- $S \cap \mathcal{S} = \emptyset$,
- *if $g\,h \in S$, then $g \in S$ or $h \in S$,*
- *if $\mathsf{hash}(h) \in S$, then $h \in S$,*

Then $I_k[S]$ is honest.

Proof sketch: A case analysis on the types of adversary strands shows that entry points to $I_k[S]$ cannot be on any adversary strand but those allowed by Definition 11. (A fuller example of an analagous proof is that of Theorem 6.11 in [12].) ■

Theorem 2 *If $K \in \mathcal{K}_{Sym}$ is never the term of a node in $\mathcal{B}$, then for any $h \in \mathcal{A}$ the term $\{|h|\}^r_K$ must originate on a regular strand.*

Proof: Suppose that $\{|h|\}^r_K$ originates on a adversary strand. By examining the forms of adversary strands, we see that the only strand on which it can originate is a **E** strand. But in that case, there is a previous node on that strand with K as its term, a contradiction. ■

Theorem 3 *If $K \in \mathcal{K}_{hash}$ is never the term of a node in $\mathcal{B}$, then for any $h \in \mathcal{A}$ the term $\{|h|\}^r_K$ must originate on a regular strand.*

Proof: The same as that of Theorem 2. ■

Theorem 4 *If $K_s \in \mathcal{K}_{Sig}$ is never the term of a node in $\mathcal{B}$, then for any $h \in \mathcal{A}$ the term $[h]^r_K$ must originate on a regular strand.*

Proof: Suppose that $[h]^r_{K_s}$ originates on a adversary strand. By examining the forms of adversary strands, we see that the only strand on which it can originate is a σ strand. But in that case, there is a previous node on that strand with K_s as its term, a contradiction. ■

Now that we have general theorems, we can apply these to the protocol from the beginning of the paper:

Definition 13 *Let $\mathtt{Cl}[C, S, d_1, d_2]$ be the set of strands of the form:*

$$\begin{array}{lcl} \langle & + & C, \\ & - & S\ [d_1]^{r_1}_{K_S}, \\ & + & [d_2]^{r_2}_{K_C}\ \{|T_1\,C\,S|\}^{r_3}_{K'} \\ & - & \{|T_2\,C\,S|\}^{r_4}_{K'} \end{array}$$

for some $r_1, r_2, r_3, r_4 \in \mathcal{R}$, where $K' = \mathsf{hash}(\mathsf{DH}(d_1, d_2))$

Definition 14 *Let $\mathtt{Sv}[C, S, d_1, d_2]$ be $\mathtt{Cl}[C, S, d_1, d_2]$ with all the signs reversed.*

First we prove that authentication from server to client is assured. That is, we show that if a client terminates its execution of the protocol successfully (the bundle has an entire $\mathtt{Cl}[C, S, d_1, d_2]$ strand) then the server finishes a corresponding run of the protocol (the bundle has contains an entire $\mathtt{Sv}[C, S, d_1, d_2]$ strand):

Theorem 5 *Let $\mathcal{B}$ be a bundle containing the strands of Definitions 7, 13 and 14. Suppose that d_1, $d_2 \notin \mathcal{D}_P$ and uniquely arise, that K_s, $K' \notin \mathcal{K}_\mathcal{P}$, and that the Diffie-Hellman problem is hard. Then if $\mathcal{B}$ contains some strand in $\mathtt{Cl}[C, S, d_1, d_2]$ of height 4, $\mathcal{B}$ must also contain some strand in $\mathtt{Sv}[C, S, d_1, d_2]$ of height 4.*

Proof: Since $K_s \notin \mathcal{K}_\mathcal{P}$ and no member of $\mathcal{K}_{Sig}$ originate on regular strands, d_1 must originate on a regular strand (Theorem 4). Hence, both d_1 and d_2 uniquely arise on regular strands. Since the protocol of Definitions 13 and 14 is both silent and conservative, $\mathsf{DH}(1d_1, d_2)$ never originates in $\mathcal{B}$ (Theorem 9). Since $K' = \mathsf{hash}(\mathsf{DH}(d_1, d_2)) \notin \mathcal{K}_\mathcal{P}$, K' is never the term of a node in $\mathcal{B}$, and so $\{|T_2\,C\,S|\}^{r_4}_{K'}$ must originate on a regular strand (Theorem 2). By inspection, it must be node 4 of $\mathtt{Sv}[C, S, d_1, d_2]$. ■

Now we show the corresponding theorem: that if the server finished a run of the protocol (the bundle has contains an entire $\mathtt{Sv}[C, S, d_1, d_2]$ strand) then the client must have finished almost all of a corresponding run (the bundle has almost an entire $\mathtt{Cl}[C, S, d_1, d_2]$ strand). We cannot guarantee that the client finishes the run since the server has no way of knowing that the last message of the protocol actually arrives:

Theorem 6 *Let $\mathcal{B}$ be a bundle containing the strands of Definitions 7, 13, and 13. Suppose that d_1, $d_2 \notin \mathcal{D}_P$ and uniquely arise, that K_c, $K' \notin \mathcal{K}_\mathcal{P}$, and that the Diffie-Hellman problem is hard. Then if $\mathcal{B}$ contains some strand in $\mathtt{Sv}[C, S, d_1, d_2]$ of height 4, $\mathcal{B}$ must also contain some strand in $\mathtt{Cl}[C, S, d_1, d_2]$ of height 3.*

Proof: Since $K_c \notin \mathcal{K}_\mathcal{P}$ and no member of $\mathcal{K}_{Sig}$ originate on regular strands, d_2 must originate on a regular strand (Theorem 4). Hence, both d_1 and d_2 uniquely arise on regular strands. Since the protocol of Definitions 13 and 14 is both silent and conservative, $\mathsf{DH}(1d_1, d_2)$ never originates in $\mathcal{B}$ (Theorem 9). Since $K' = \mathsf{hash}(\mathsf{DH}(d_1, d_2)) \notin \mathcal{K}_\mathcal{P}$, K' is never the term of a node in $\mathcal{B}$, and so $\{|T_1\,C\,S|\}^{r_3}_{K'}$ must originate on a regular strand (Theorem 2). By inspection, it must be node 3 of $\mathtt{Cl}[C, S, d_1, d_2]$. ■

As can be seen, the global property of Definition 10 leads to very short and simple proofs for a natural class of protocols. In the next section, we give the promised formal justification of this property.

6 Diffie-Hellman in Strand Spaces, Formally

The main idea behind our justification is that there is a natural conversion from attacks in the formal setting (bundles, in particular) to computational algorithms. We give this conversion in this section, and show that if the bundle violates the global property then the resulting algorithm solves the Diffie-Hellman problem.

This is not enough, however. To violate the Diffie-Hellman assumption, we need an algorithm that both solves the Diffie-Hellman problem and is *efficient*. Since our conversion makes no assumptions about the forms of regular strands, there are no guarantees about the complexity of the resulting algorithm. In the next section, we show that if the regular strands are silent and conservative, then the resulting algorithm will be efficient. (Or rather, a slight variant of the resulting algorithm that *simulates* the regular strands will be efficient.)

6.1 Relating Bundles and Computational Algorithms

A word about how the mapping from bundles to algorithms will proceed: the resulting algorithm will calculate (and store in a table) a value for each node in the bundle by recursing on both the structure of the bundle and the structure of each term. The recursion along the bundle structure is relatively straightforward: early nodes are calculated before later ones. The recursion along message structure, on the other hand, presents an interesting issue: how should the algorithm build values for compound messages from those for atomic ones?

The algorithm will use, as black boxes, computational algorithms for encryption, signing, and hashing. (These algorithms are defined in Appendix A.) The mapping itself assumes no properties about these sub-algorithms, meaning we are free to choose these sub-algorithms arbitrarily. However, this freedom is short-lived: for efficiency conditions, we will later (Section 6.2) need to assume that one of these algorithms meets a standard definition of security.

Given a formal bundle $\mathcal{B}$, we will map it onto an algorithm $\mathtt{A}_{\mathcal{B}}$ in the following way:

Definition 15 *Let* $(\mathsf{G_e}, \mathsf{E}, \mathsf{D})$ *be an encryption scheme,* $(\mathsf{G_s}, \mathsf{S}, \mathsf{V})$ *be a signature scheme, and* $(\mathsf{G_h}, \mathsf{H})$ *be a hash scheme. Let* $\mathcal{B}$ *be a bundle over* $\mathcal{A}$*. Then* $\mathtt{A}_{\mathcal{B}}(1^\eta)$ *is the following algorithm:*

- *First, a table* T *is created to map elements of* $\mathcal{A}$ *to bit-strings. We assume that this mapping to be consistent over the entire bundle. For example, every instance of* $K \in \mathcal{K}$ *that occurs in* $\mathcal{B}$ *is intended to "represent" the same bit-string throughout the strand. At the beginning of the execution, this table is empty.*
- *A group* $G_p \leftarrow \mathtt{IG}(1^\eta)$ *is generated for Diffie-Hellman.*
- *A hash function* $h \leftarrow \mathsf{G_h}(1^\eta)$ *is generated for hashing.*
- *Each node in the bundle is then replaced with a bit-string value, starting with minimal nodes and working forward. That is, a node* n *is replaced with a bit-string value* v_n *only after every* $n' \leq_{\mathcal{B}} n$*. The exact manner in which a bit-string is chosen for* n *depends on the sign of* n *and the type of strand on which it lies:*
- *Suppose that* n *is a positive node* $+d_k \in \mathcal{D}$ *and it lies upon an* f *strand. Then:*
  - f *is a PPT-computable function, and*
  - *By inductive hypothesis, values* $v_1, v_2, \ldots v_{k-1}$ *have already been chosen for the nodes* $-d_1$, $-d_2, \ldots -d_{k-1}$ *previous on the strand.*

  The value v_n *for* $+d_k$ *is chosen by running* $\mathtt{M}_f(\langle v_1, v_2, \ldots v_{k-1}\rangle)$ *and returning the output. Additionally, this value is stored for* d_k *in the table* T *if no value for* d_k *is already present there.*
- *If* n *is a positive node* $+M$ *and it lies upon any kind of strand other than an* f *strand, then there are two cases:*
  - *If there exists a bit-string* m *in the table* T *as the value for* M*, then* m *is the value returned.*
  - *If* M *is not in the table* T*, it generates a value* v *for* M*. The value* v *is then stored in* T *and returned. The value* v *is generated by recursing on the structure of* M*:*
    - *If* M *is an atomic message, then the value* v *is chosen in some appropriate way. For keys (which are not in* $\mathcal{K}_{hash}$*) the appropriate key generation algorithm is run. For randomness, random strings of length* $Q(\eta)$ *are chosen uniformly from* $\{0,1\}^{Q(\eta)}$*. (The polynomial* Q *here is the same as that in Appendix A.) Diffie-Hellman values (in* $\mathcal{D} \setminus \mathcal{D}_{DH}$*) are created by choosing a random element* $x \leftarrow \{1, \ldots |G_p|\}$ *and calculating* g^x*. Texts are converted into bit-strings in some*

arbitrary way.[10] *The value is put into the table T as the value for M and returned. (In the case of signature keys, both the signing and verfication keys are stored in T.)*

- *If $M = \mathsf{hash}(M')$ then the algorithm recursively gets a value m' for M'. It then sets $v = \mathsf{G_e}(1^\eta, h(m'))$. (Note that we are now considering $\mathsf{G_e}$ to be a deterministic algorithm of two inputs, and using the output of the hash as the second, random, input.)*
- *If $M = M_1\, M_2$, then the algorithm recursively gets values m_1 for M_1 and m_2 for M_2. It returns $v = \langle m_1, m_1 \rangle$.*
- *If $M = \{\!|M'|\!\}_K^R$ or $[M']_K^R$, then the algorithm recursively gets values m' for M', r for R and k for K. It then calculates $v = \mathsf{E}(m', r, k)$ or $v = \mathsf{S}(m', k'r)$, respectively.*
- *If $M = \mathsf{DH}(d_1, d_2)$ then the algorithm recursively gets values g^x for d_1 and g^y for d_2. It calculates $v = g^{xy}$. (Again, we note that calculating g^{xy} from g^x and g^y may not be efficiently computable, but delay discussion of this issue until the next section.)*

- *If n is a negative node $-M$ (on a strand s) then there exists in the bundle a node $+M$ so that $+M \rightarrow -M$. By assumption, a value m' has already be assigned to the node $+M$, which means that a value v has been assigned to M in T. We accept v for $-M$ also.*

The above algorithm converts each node of the bundle $\mathcal{B}$ into a bit-string. We now define what it means for it to have performed the conversion correctly:

Definition 16 *Let $\mathcal{B}$ be a bundle and $\mathtt{A}_\mathcal{B}$ be the algorithm derived from $\mathcal{B}$ as per Definition 15. Then an execution of $\mathtt{A}_\mathcal{B}$ is "correct" when, for every f strand in the bundle and every execution of $\mathtt{M}_f$, if $\mathtt{M}_f$ is run on $\langle x_1, x_2, \ldots x_n \rangle$ it outputs $f(x_1, x_2, \ldots x_n)$.*

Theorem 7 *If $\mathcal{B}$ is a bundle, then*

$$\Pr\left[\mathtt{A}_\mathcal{B}(\eta) \textit{ computes properly }\right] \geq \frac{1}{q(\eta)}$$

for some polynomial q.

Proof: If every execution of $\mathtt{M}_f$ properly calculates f (for every f strand in the bundle) then the algorithm $\mathtt{A}_\mathcal{B}$ properly executes. What are the odds that each execution of $\mathtt{M}_f(x)$ properly calculates $f(x)$? By definition, the probability of a successful calculation is polynomial in $|x|$. During the execution of $\mathtt{A}_\mathcal{B}$, $\mathtt{M}_f$ will only be executed on the encodings of terms. Encodings of atomic terms are of length polynomial in η by definition, or are generated by algorithms that run in time polynomial in η. Furthermore, the encoding of a compound term is generated by a polynomial time algorithm running on the encodings of terms. Since the "depth" (or structure) of a term is constant with respect to η, it must be that all encodings of terms have length polynomial in η.

Lastly, we assume that the random coinflips for each execution of $\mathtt{M}_f$ are independent, and note that the number of f strands in $\mathcal{B}$ is constant with respect to η. Hence, the odds that all executions of $\mathtt{M}_f$ properly calculate f (and hence that $\mathtt{A}_\mathcal{B}$ executes properly) is the product of a constant number of probabilities, all of which are larger than a polynomial in η. Hence, the probability that $\mathtt{A}_\mathcal{B}$ executes properly is larger than some polynomial in η. ∎

We note that this result is independent of the choices for encryption, signature, and hash algorithms. This is because the operations available to the adversary with regards to these schemes are deterministic (once the random input is fixed). The only probability comes in the form of f-strands, and each of those are assumed to be computable in PPT time. Hence, any bundle can be computed with non-negligible probability.

Theorem 8 *Suppose $\mathtt{A}_\mathcal{B}$ correctly executes, and let T be the table at the end of the execution. For all d_1, $d_2 \in \mathcal{D}$, if $T(d_1) = g^x$ and $T(d_2) = g^y$, then $T(\mathsf{DH}(d_1, d_2)) = g^{xy}$.*

Proof: Consider where in the bundle the value $\mathsf{DH}(d_1, d_2)$ arise. If it only arises on regular strands, then it will be assigned the value g^{xy}. If it arises on an f strand, then

$$\mathsf{DH}(d_1, d_2) = f(M_1, M_2, \ldots M_n)$$

where each M_i is some message. It may be that $M_i = d_i$ for some $d_i \in \mathcal{D}$, which may also arise on an f' strand. Hence, it may be that

$$\mathsf{DH}(d_1, d_2) = f(M_1, \ldots, f'(N_1, \ldots N_m) \ldots, M_n)$$

and so on. But the table T may not contain the correct evaluations of f, f' and so on. The values in T are created by running the machines M_f, $M_{f'}$ and so on. But if the algorithm correctly executes then each run of the machine M_f correctly evaluates f, and the same for $M_{f'}$ and so on. Hence, if every f strand is calculated correctly, then the value for $\mathsf{DH}(d_1, d_2)$ in T will be evaluated correctly, which gives it the value of g^{xy}. ∎

Hence, if a bundle uses $\mathsf{DH}(d_1, d_2)$ at any point, then the algorithm can be used to solve the Diffie-Hellman problem. However, the algorithm $\mathtt{A}_\mathcal{B}$ may not be computable

[10] So long as the mapping from formal texts to finite bit-strings is efficiently computable and deterministic, it does not matter how the translation is actually done.

in probabilistic polynomial time. The algorithm as described requires that the Diffie-Hellman problem be solved for each term $\mathsf{DH}(d_1, d_2)$ in the bundle that doesn't arise on an f-strand. That is, if the bundle contains an instance of $\mathsf{DH}(d_1, d_2)$ that arises on a regular strand, the resulting algorithm may be forced to solve Diffie-Hellman. In the next section, we discuss a way around this difficulty.

6.2 Efficiency Concerns and Simulation

Assume that there exists a bundle where the regular strands are silent and conservative and which violates security property $\mathcal{DH}$. As shown in the previous section, this bundle maps to an algorithm that solves the Diffie-Hellman problem. However, the algorithm may not be efficient. In particular, the algorithm assigns the value g^{xy} to the formal term $\mathsf{DH}(d_1, d_2)$ (when it also assigns g^x to d_1 and g^y to d_2). This may require the algorithm to solve the Diffie-Hellman problem directly — an operation we are explicitly assuming to be inefficient.

However, all adversary strands *are* efficiently computable; the algorithm would only need to solve Diffie-Hellman when calculating the values on regular strands. Furthermore, we now assume that all of the regular strands are conservative, and hence Diffie-Hellman values are only used on regular strands to make keys. Since we assume that making a key from a Diffie-Hellman value involves hashing it first, we can use this to avoid an infeasible computation.

The central idea is that a hash algorithm can be simulated by a random function. In particular, we assume that the hash function is pseudorandom:

Definition 17 *Let $\{R_\eta\}$ be a family of function families with the following two properties:*

1. $\forall r \in R_\eta$, *r is a function from $\{0,1\}^*$ to $\{0,1\}^{Q(\eta)}$*
2. $\forall \eta \in \mathsf{Parameter}, \forall s \in \{0,1\}^*, \forall t \in \{0,1\}^{Q(\eta)}$,
$$\Pr\left[r \leftarrow R_\eta : r(s) = t\right] = 2^{-Q(\eta)}$$

Then $\{R_\eta\}$ is a random function family.

Definition 18 *A hash function is* pseudorandom *if for all PPT distinguishers $\mathtt{A}$, for all polynomials q, and for all sufficiently large η:*

$$\left|\Pr\left[r \leftarrow R_\eta : \mathtt{A}^{r(\cdot)}(1^\eta) = 1\right] - \Pr\left[h \leftarrow \mathsf{G_h}(1^\eta) : \mathtt{A}^{h(\cdot)}(1^\eta) = 1\right]\right| \leq \frac{1}{q(\eta)}$$

That is, a hash algorithm scheme is pseudo-random if a randomly-chosen hash operation is indistinguishable from a random function. Hence, if we assume that regular participants make keys from Diffie-Hellman values by hashing them with a randomly-chosen hash operation, we can simply choose random values instead. If this modification changes the output distribution of the resulting algorithm, then the hash algorithm is not, in fact, pseudorandom. (The fact that the regular strands are silent is essential here, as we will see later in the proof.)

Theorem 9 *Suppose that the protocol Π is both silent and conservative with respect to Diffie-Hellman. Suppose that exists a pseudorandom hash algorithm $(\mathsf{G_h}, \mathsf{H})$ and all functions f with f-strands in $\mathcal{B}$ are probabilistic polynomial time-computable with respect to $(\mathsf{G_h}, \mathsf{H})$. If there exists a bundle $\mathcal{B}$ over a Π which violates the formal Diffie-Hellman property $\mathcal{DH}$, then the computational Diffie-Hellman assumption is false.*

Proof: By assumption, $\mathcal{B}$ violates security property $\mathcal{DH}$. Then:

- d_1 and d_2 arise only on regular strands in $\mathcal{B}$,
- $\mathsf{DH}(d_1, d_2)$ never originates on a regular node, and
- $\mathsf{DH}(d_1, d_2)$ originates in $\mathcal{B}$.

Let n be a *minimal* origination point of $\mathsf{DH}(d_1, d_2)$ in $\mathcal{B}$. Note that all origination points of $\mathsf{DH}(d_1, d_2)$ are on adversary strands, including n. By inspection of the form of adversary strands, it must be that the term of n is in fact $\mathsf{DH}(d_1, d_2)$ itself. (If it contained $\mathsf{DH}(d_1, d_2)$ as a subterm, then $\mathsf{DH}(d_1, d_2)$ must be a subterm of a previous node on that strand, and n would not be an origination point.)

Let $\mathcal{B}|_n$ be the set of all nodes in $\mathcal{B}$ which are "before" n. That is, let
$$\mathcal{B}|_n = \{n' | n' \leq_\mathcal{B} n\}$$

We construct an adversary $\mathtt{A}$ that breaks the computational Diffie-Hellman assumption in the following way: $\mathtt{A}(p, g, g^x, g^y)$ simulates $\mathtt{A}_{\mathcal{B}|_n}$, with the following important exceptions:

1. Instead of generating its own group $G_p \leftarrow \mathtt{IG}(\eta)$, $\mathtt{A}$ will use the group specified by its first input p and use its second input g as a generator.
2. Instead of the table T being empty at initialization, it contains an entry mapping d_1 to g^x and an entry mapping d_2 to g^y.
3. When $\mathtt{A}_{\mathcal{B}|_n}$ calculates a value for $\mathsf{DH}(d_a, d_b)$, it seems to need to solve the Diffie-Hellman problem in order to do so. However, we can avoid this calculation by considering the kinds of nodes which would cause $\mathtt{A}_{\mathcal{B}|_n}$ to calculate a value for $\mathsf{DH}(d_a, d_b)$.

- It could be an f strand, in which case $\mathtt{A}$ simulates $\mathtt{M}_f$ as $\mathtt{A}_{\mathcal{B}|_n}$ would.
- If could be a regular strand, in which case we know that $\mathsf{DH}(d_a, d_b)$ is not a subterm of the node in question. (If it were, then it would have originated there.) Let the node in question be $+M$. Since it is an ingredient of the term but not a subterm, we can see by examining the term structure that $\mathsf{DH}(d_a, d_b) \preceq \mathsf{hash}(N) \preceq M$. Therefore, we only need to calculate a Diffie-Hellman value as part of computing a hash. Since the hash is pseudorandom, we employ a trick: instead of calculating the hash, we return a random value instead. That is, instead of calculating M normally, $\mathtt{A}$ chooses $n' \leftarrow \{0,1\}^{q(\eta)}$. It returns n' for $\mathsf{hash}(N)$, and store n' in the table T as the value for $\mathsf{hash}(N)$. It does not calculate a value for N or any of its ingredients (including $\mathsf{DH}(d_a, d_b)$) as part of the calculation of a value for M.

When finished simulating $\mathtt{A}_{\mathcal{B}|_n}$, the adversary $\mathtt{A}$ selects the value g^z calculated for the node n, and returns g^z as its output.

What is the likelihood that the new algorithm $\mathtt{A}$ will output the correct value?

Let us revisit the original algorithm $\mathtt{A}_{\mathcal{B}|_n}$. We know from Theorem 7 that for some polynomial q:

$$\Pr\left[\mathtt{A}_{\mathcal{B}|_n}(\eta) \text{ computes properly}\right] \geq \frac{1}{q(\eta)}.$$

Note that we can modify $\mathtt{A}_{\mathcal{B}|_n}$ to take the group, the generator, and the values for d_1 and d_2 as inputs. In that case, running the new algorithm on random inputs is exactly the same as running it before the modifications:

$$\Pr[\ \begin{array}{l}\langle p, g\rangle \leftarrow \mathtt{IG}(1^\eta);\\ x, y \leftarrow \{1, 2, \ldots, |G_p|\} :\\ \mathtt{A}_{\mathcal{B}|_n}(p, g, g^x, g^y) \text{ computes properly}\end{array}\ ] \geq \tfrac{1}{q(\eta)}$$

We can also modify $\mathtt{A}_{\mathcal{B}|_n}$ to output g^z, where $\langle g^z\rangle$ is the value computed for node n. Due to the definition of proper computation:

$$\Pr[\ \begin{array}{l}\langle p, g\rangle \leftarrow \mathtt{IG}(1^\eta);\\ x, y \leftarrow \{1, 2, \ldots, |G_p|\};\\ g^z \leftarrow \mathtt{A}_{\mathcal{B}|_n}(p, g, g^x, g^y) :\\ g^{xy} = g^z\end{array}\ ] \geq \tfrac{1}{q(\eta)}$$

That is, the original algorithm $\mathtt{A}_{\mathcal{B}|_n}$ can calculate the Diffie-Hellman value g^{xy} with some polynomial probability. But we are running $\mathtt{A}$, not $\mathtt{A}_{\mathcal{B}|_n}$. Will the new algorithm have the same advantage? It is not clear: $\mathtt{A}$ uses random values for hashing, while the original algorithm $\mathtt{A}_{\mathcal{B}|_n}$ calculates the values by application of the hash algorithm. However, suppose that the probability of success for $\mathtt{A}$ were negligible while the probability of success for $\mathtt{A}_{\mathcal{B}|_n}$ is non non-negligible. That is, let P_1^η be the probability:

$$\Pr[\ \begin{array}{l}\langle p, g\rangle \leftarrow \mathtt{IG}(1^n);\\ x, y \leftarrow \{1, 2, \ldots, |G_p|\};\\ g^z \leftarrow \mathtt{A}_{\mathcal{B}|_n}(p, g, g^x, g^y) :\\ g^{xy} = g^z\end{array}\ ]$$

and P_2^η be the probability:

$$\Pr[\ \begin{array}{l}\langle p, g\rangle \leftarrow \mathtt{IG}(1^n);\\ x, y \leftarrow \{1, 2, \ldots, |G_p|\};\\ g^z \leftarrow \mathtt{A}(p, g, g^x, g^y) :\\ g^{xy} = g^z\end{array}\ ]$$

If P_1 is non-negligible and P_2 is negligible, then

$$P_1^\eta - P_2^\eta = |P_1^\eta - P_2^\eta| \geq \frac{1}{q'(\eta)}$$

Note that the only difference between the two experiments is in how hashes are calculated. In P_1^η, hashes are calculated by actually calculating the pre-image of the hash, then taking the hash under a randomly chosen hash function. In the second probability P_2^η, hashes are taken by returning random values. If these probabilities are non-negligibly different, then we can distinguish the hash scheme from a random function family. Let the distinguisher be:

$\mathtt{D}^{g(\cdot)}(\eta) =$

1. Choose random $\langle p, g\rangle \leftarrow \mathtt{IG}(1^n)$ and x, $y \leftarrow \{1, 2, \ldots, |G_p|\}$.
2. Simulate $\mathtt{A}(p, g, g^x, g^y)$ with one difference: instead of calculating hash values by any calculation, use the oracle $g(\cdot)$. Note that now the algorithm $\mathtt{D}$ knows the exponents of g^x and g^y, and so knows the exponent for every Diffie-Hellman value that arises on a regular strand. Since the protocol is conservative, whenever $\mathtt{D}$ needs to calculate a value for $\mathsf{DH}(d_a, d_b)$ it is the case that either d_a or d_b arises on a regular strand. Hence, whenever $\mathtt{D}$ needs to perform the Diffie-Hellman operation as part of the simulation, it knows one of the two relevant exponents and the calculation is easy.
3. When the simulation returns g^z, test to see if $g^z = g^{xy}$. (Since $\mathtt{D}$ chose x and y, it can perform this test.) If it does, return 1. Otherwise return 0.

In other words, this algorithm also creates a value for each node in $\mathcal{B}|_n$. However, since it knows the exponent for every Diffie-Hellman value d, it can calculate the value for $\mathsf{DH}(d, d')$ efficiently.

Since this distinguisher returns 1 with probability P_1^η if g is a random function and with probability P_1^η if g is a randomly chosen hash function, this can distinguish the hash from random. Since the hash family is pseudorandom, we know this cannot be.

Hence, the advantage of the original algorithm $\mathtt{A}_{\mathcal{B}_n}$ and the advantage of the new algorithm $\mathtt{A}$ cannot differ by a polynomial fraction. That is, for all polynomials q', for sufficiently large η,

$$|P_1^\eta - P_2^\eta| \leq \frac{1}{q'(\eta)}.$$

Hence, for all polynomials q', for sufficiently large η,

$$P_2^\eta \geq \frac{1}{q(\eta)} - \frac{1}{q'(\eta)}$$

or by letting $q' = 2q$,

$$P_2^\eta \geq \frac{1}{2q(\eta)}.$$

■

In other words, suppose that the hash function is pseudorandom, and that the adversary strands are all polynomial-time computable over the choices of hash, encryption, and signature functions. Then if there exists a bundle over a conservative, silent protocol that violates the security property $\mathcal{DH}$ in Definition 10, the computational Diffie-Hellman assumption is false over the group family in question. Conversely, if the Diffie-Hellman problem is hard over the group, then there can be no silent and conservative bundle which violates the security condition $\mathcal{DH}$.

7 Conclusion

The primary purpose of this work is two-fold:

1. To allow the Strand Space method to analyze protocols that use the Diffie-Hellman key exchange, and
2. To show how the computational model can be used to define and/or justify new security assumptions in the formal model.

To this end, we have formalized a security condition that summarizes — in a form appropriate for Strand Spaces — the security provided by the Diffie-Hellman assumption. To justify this condition, we provided a method for transforming bundles into computational algorithms. Under reasonable assumptions on the form of the protocol and the underlying cryptography, our condition can be violated only if the computation Diffie-Hellman assumption is false.

We believe this represents a new step in the development of formal cryptographic analysis. In particular, we believe this to be the first effort to use the computational model to incorporate Diffie-Hellman into the formal model. Previous work on protocols that use Diffie-Hellman [10] have focused on finding attacks rather than proving security. Hence, they have made simplifying assumptions on the powers of the adversary and the nature of possible attacks. (In particular, they assume that the only way to solve the Diffie-Hellman problem is to solve the corresponding Discrete Log problem — a simplifying assumption much like those made about encryption.) Our work is focused on proofs of security, and so we assume only what can be justified in terms of computational cryptography. Hence, proofs in our framework will be as strong as the Diffie-Hellman assumption. (Note, however, our work is not as widely applicable as that in [10]: we cannot yet consider common group-keying protocols, for example.)

On the other hand, there have been many interesting papers that connect the formal (i. e. Dolev-Yao) model with the computational approach[1, 2, 3, 8, 9]. However, these papers focus on long-standing simplifications and abstractions Our work is novel in that it used the computational approach to derive and justify *new* abstractions.

We would like to see this work continue in two ways. First, we would like to see our security condition translated to settings other than Strand Spaces. Furthermore, we hope that the security condition can be used to analyze real-world protocols, or even be used to help design new ones.

Second, we would like to see if the assumptions of this paper can be weakened. Our assumptions regarding the underlying cryptography are quite weak: only that the hashing is pseudorandom. However, this weakness on the cryptography is balanced by the strength of the formal assumptions (Definitions 8 and 9). Because these assumptions are so strong, there are very likely secure protocols which cannot yet be proven secure in our framework. We would be interested to see if these conditions could be weakened. Alternately, there may be other conditions which guarantee the simulatability of the regular strands. If so, we would be interested in seeing them, and would be particularly interested in knowing if there are necessary and/or sufficient conditions for simulatability.

Lastly, we believe that the main technique of this paper to be novel and highly applicable. We would very much like to see it used to incorporate other primitives into formal models.

A Computational Primitives

Here, we define the computational algorithms used in the mapping of Section 6. First, some helper sets:

Definition 19 *We use the following definitions:*

- $\mathsf{Parameter} = \mathcal{N}$
- $\mathsf{Coins} : \mathsf{Parameter} \to \mathcal{P}(\{0,1\}^*)$
- $\mathsf{String} = \{0,1\}^*$

We define Coins, the set of random strings to be a function of the security parameter because the number of coinflips used by the cryptographic primitives grow with the security parameter. In general, we will assume that for all $\eta \in \mathsf{Parameter}$,

$$\mathsf{Coins}(\eta) = \{0,1\}^{Q(\eta)}$$

for some polynomial Q.

Definition 20 *An symmetric encryption scheme [7] is a triple of algorithms* $(\mathsf{G_e}, \mathsf{E}, \mathsf{D})$*:*

- $\mathsf{G_e} : \mathsf{Parameter} \times \mathsf{Coins} \to \mathsf{SymmetricKey}$ *is the (randomized) key generation algorithm,*
- $\mathsf{E} : \mathsf{Plaintext} \times \mathsf{Coins} \times \mathsf{SymmetricKey} \to \mathsf{Ciphertext}$ *is the (randomized) encryption algorithm, and*
- $\mathsf{D} : \mathsf{String} \times \mathsf{SymmetricKey} \to \mathsf{Plaintext} \cup \{\perp\}$ *is the decryption algorithm, which we assume returns* $\perp$ *whenever the input string is not a valid encryption under the given key.*

SymmetricKey, Plaintext *and* Ciphertext *vary between encryption algorithms and implicitly depend on the parameter. It is required that for any message length* i*,* Plaintext *contains either all messages of length* i *or none of them. Also, it is required that for all* $r \in \mathsf{Coins}(\eta)$*, all* k *generated by* $\mathsf{G_e}(1^\eta, r)$*, and all* $m \in \mathsf{Plaintext}(\eta)$*,*

$$\mathsf{D}(\mathsf{E}(m, r, k), k) = m.$$

Definition 21 *A* digital signature scheme *is a triple of algorithms:* $(\mathsf{G_s}, \mathsf{S}, \mathsf{V})$*:*

- $\mathsf{G_s} : \mathsf{Parameter} \times \mathsf{Coins} \to \mathsf{SignatureKeys} \times \mathsf{VerificationKeys}$ *is the (randomized) key generation algorithm,*
- $\mathsf{S} : \mathsf{String} \times \mathsf{Coins} \times \mathsf{SignatureKeys} \to \mathsf{Signatures}$ *is the (randomized) signature algorithm, and*
- $\mathsf{V} : \mathsf{String} \times \mathsf{Signatures} \times \mathsf{VerificationKeys} \to \{0,1\}$ *is the verification algorithm*

It is required that for all $r \in \mathsf{Coins}(\eta)$*, all* $\langle k, k^{-1} \rangle$ *generated by* $\mathsf{G_s}(1^\eta, r)$*, and all* $m \in \mathsf{String}(\eta)$*,*

$$\mathsf{V}(m, \mathsf{S}(m, r, k), k^{-1}) = 1.$$

Definition 22 *A* hash algorithm *is a pair of algorithms* $(\mathsf{G_h}, \mathsf{H})$*, where:*

- $\mathsf{G_h} : \mathsf{Parameter} \times \mathsf{Coins} \to \mathsf{HashFunctions}$ *generates hash functions (and is randomized), and*
- $\mathsf{H} : \mathsf{HashFunctions} \times \mathsf{String} \to \mathsf{String}$ *evaluates the hash function.*

For every random string $r \in \mathsf{Coins}(\eta)$*, all strings* $s \in \{0,1\}^*$*,* $\mathsf{H}(\mathsf{G_h}(1^\eta, r), s)$ *will be a string of length* $Q(\eta)$*.*

We will write $\mathsf{G_h}(1^\eta)$ for the probability distribution induced by $\mathsf{G_h}(1^\eta, r)$ where r is chosen randomly (uniformly) from $\mathsf{Coins}(\eta)$.

References

[1] Martín Abadi and Jan Jürjens. Formal eavesdropping and its computational interpretation. *Lecture Notes in Computer Science*, 2215:82ff., 2001.

[2] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[3] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A universally composable cryptographic library. Available at `http://eprint.iacr.org/2003/015/`, January 2003.

[4] Dan Boneh. The decision Diffie–Hellman problem. In *Proceedings of the Third Algorithmic Number Theory Symposium*, number 1423 in Lecture Notes in Computer Science, pages 48–63. Springer–Verlag, 1998.

[5] T. Dierks and C. Allen. The TLS protocol. RFC 2246, January 1999.

[6] Daniel Dolev and Andrew Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 29:198–208, 1983.

[7] Shafi Goldwasser and Mihir Bellare. Lecture notes on cryptography. Avaliable at http://www.cs.ucsd.edu/users/mihir/papers/gb.html, August 1999.

[8] Joshua D. Guttman, F. Javier Thayer, and Lenore D. Zuck. The faithfulness of abstract protocol analysis: Message authentication. *Journal of Computer Security*, 2003. Forthcoming.

[9] Jonathan Herzog. Computational soundness of formal adversaries. Master's thesis, Massachusetts Institute of Technology, 2002.

[10] Olivier Pereira and Jean-Jacques Quisquater. A security analysis of the cliques protocols suites. In *14th IEEE Computer Security Foundations Workshop — CSFW'01*, pages 73–81, Cape Breton, Canada, 11–13 June 2001. IEEE Computer Society Press.

[11] F. Javier THAYER Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Mixed strand spaces. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1999.

[12] F. Javier THAYER Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2/3):191–230, 1999.

[13] T. Ylonen, T. Kivinen, and M. Saarinen. SSH transport layer prototcol. Internet draft, November 1997. Also named draft-ietf-secsh-transport-01.txt.