Plug and Play Encryption

Donald Beaver *

IBM/Transarc

Abstract. We present a novel protocol for secret key exchange that is provably secure against attacks by an adversary that is free to attack zero, one, or both parties in an adaptive fashion, at any time. This high degree of robustness enables larger, multiparty interactions (including multiparty secure computations) to substitute our protocol for secure private channels in a simple, plug-and-play fashion, without simultaneously limiting security analysis to attacks by static adversaries, *i.e.* adversaries whose corruption choices are fixed in advance.
No reliance on the assistance of third parties or on erasing partial computations is required. In addition to providing order-of-magnitude speedups over alternative approaches, the simplicity of our protocols lends itself to simple demonstrations of security. We present constructions that are based on a novel and counterintuitive use of the Diffie-Hellman key exchange protocol; our methods extend to other standard cryptographic assumptions as well.

1 Introduction

Historically, the pressing theme of cryptology has been to convey a private message securely, as though an absolutely secure private channel were available. Encryption schemes of ever increasing robustness have been proposed, from private-key methods that assume an initially-secure exchange, to public-key schemes that enable key exchange over public lines without prior communication [DH76]. Although not always explicit, one goal is common to all such efforts: to be able to plug in a replacement for an absolutely secure channel without compromising security, *i.e.* while maintaining the level of security of the original (even if imaginary) channel.

The construction of large systems is guided by several natural motivations for pursuing a component-based approach. First, the design and construction of large systems is simplified, as is the analysis of their properties. Second, the ability to replace costly or idealized components reduces the overall costs of implementing a system. Third, the system flexibly accommodates advances in quality and technology – as well as newly uncovered disadvantages or flaws. The goal of component-based design is thus not merely aesthetic: functionality, utility, affordability, and flexibility tend to decrease as complexity increases.

* Transarc Corp., Pittsburgh, PA 15219; 412-338-4365; beaver@transarc.com, http://www.transarc.com/~beaver.

Security is no exception. Yet security analyses often make implicit assumptions about the context in which a component is used, at the risk of compromising overall security.

In particular, when a cryptosystem is analyzed in isolation, *e.g.* as a system involving just two parties and an eavesdropper, many constraints that apply to its use in a larger system are easily overlooked, or misleadingly treated as moot. Even if the result is not catastrophic failure, there is a price: the deceptively simple verification of security in isolation provides no formal guarantee about security after installation.

The goal of this work is to provide *plug-and-play encryption systems* that are robust, sufficiently efficient and (most importantly) sufficiently analyzed to merit simple installation in large-scale network interactions, whether tightly coupled (as in multiparty computations) or loosely (as on internets). In either case, we demand and achieve privacy in the face of adaptive attacks.

Adaptive Attacks. One subtle sticking point that has received relatively little attention is the distinction between static and adaptive attacks. An adversary mounting a static attack is required to choose which players it wants to corrupt before the protocol begins (although its later behavior, *viz* substitution of messages, can be freely adapted). Adaptive attacks permit the adversary to choose whom to corrupt (typically, up to some limit) at any time.

Clearly, security against adaptive attacks is the stronger, more realistic, and more desirable achievement. But analysis of static attacks is simpler, and far more common. There are indeed apparently good reasons to discard the more complex (hence more risky) analysis needed to assure adaptive security. For example, if a sender or receiver is attacked, then any messages are compromised *a fortiori*, so further analysis seems unnecessary. Worse, if a sender is attacked, then what happens when the receiver is attacked later should be moot, since any messages are likewise already compromised.

Unfortunately, this sort of reasoning takes place in a cryptographic vacuum: there exists nothing apart from sender, receiver, and eavesdropper. It does not extend to situations in which there are multiple parties employing multiple cryptosystems along with diverse other protocols and interactions. In such a setting, the behavior of the network is not made moot by the failure of two parties, hence an overall analysis *does* depend on the behavior of a component protocol when one or both of its participants are overrun.

Thus – subtly – the security of a system that has been verified only against static attacks may remain in question if it is plugged into a large-scale interaction in which attackers pick and choose victims at will.

While obvious when stated thus, this simple observation is easily overlooked when the simpler component (cryptosystem, zero-knowledge proof, *etc.*) is analyzed outside the context of the larger system.

The natural questions, then, are whether the static analysis is sufficient anyway (it is not), whether existing cryptosystems can hope to enjoy a demonstration of security in dynamic, adversarial environments (most cannot), and whether there are alternatives that can indeed be shown robust (there are).

Obstacles to Adaptive Security. The primary technical difficulty in developing cryptosystems that can be proven secure against adaptive attacks is the fact that most cryptosystems bind the sender and receiver to the cleartext message, even though the cleartext itself may be hard to calculate from the ciphertext alone.

An adversary (and anyone else) can simply observe $E(m, r)$ over a public line (where m is the cleartext and r represents random bits). Not only are the sender and receiver unable to later pretend that another message m' was sent, but the adversary can expect to discover a message m and list of random bits r that is consistent with the string $E(m, r)$, if it decides to corrupt the sender or receiver later on.

This concern arises even in the simple application of key-exchange protocols. Consider Diffie-Hellman key exchange (DH), in which Alice sends some $g^a \bmod p$, Bob responds with some $g^b \bmod p$, and the two use $K = g^{ab} \equiv (g^a)^b \equiv (g^b)^a \bmod p$ as their secret key [DH76].[2]

Unless computing discrete logarithms is feasible, an eavesdropper will likely find it hard to compute g^{ab}. But it is impossible to supply a different a', b' and $g^{a'b'}$ that are consistent with the public values g^a and g^b. Thus, it is impossible to pretend later that a value other than g^{ab} was used as the key, if the value of a or b is obtained (*e.g.* via corruption of one of the parties). Messages encrypted with g^{ab} (or some derivative) are private but immutable.

Simulation. This property, although apparently just technical, is critical in any approach that measures knowledge through Turing tests, *i.e.* through simulations [GMR89]. In the domain of computational security, zero-knowledge approaches are a common standard for demonstrating interactive security ([GMR89, GMW86, B91, MR91, B95]). Typically, one must find a simulator that presents a convincing but faked conversation without having access to the private information that normally may play a role in generating the actual conversation. If the fake conversation is indistinguishable from a real one, then we may infer that the real one leaks no "knowledge" about the sensitive private information.

In the case of encryption, this reduces to being able to simulate a ciphertext (or a key-exchange conversation) without having access to the messages and secret bits held by the sender and receiver. The static case is simple to analyze: if either party is corrupt, then the simulator is entitled to the secret message m, and can easily form a valid encryption $E(m, r)$ as necessary; if neither is corrupt, then the simulator can typically offer up $E(0, r)$ as the fake ciphertext. Because the key and random bits will never be obtained by the static adversary, distinguishing between $E(0, r)$ and $E(1, r')$ (for example) is tantamount to breaking a cryptographic assumption [GM84].

It might be argued that this simulation-based approach is unnecessarily burdensome for cryptosystems. Yet it is often the case that a larger protocol employs other convenient modules, such as zero-knowledge proofs or oblivious transfers

[2] Here, p is a large prime, g a generator of $\mathbf{Z}_p^*$, and a and b are chosen at random $\bmod(p-1)$ by the respective parties.

or bit committals, and as a result demands an overall simulator-based approach. To facilitate plug-and-play usage of cryptosystems, then, a simulator for the component cryptosystem is a minimum requirement.

Equivocation. For encryption systems, the central technical problem lies in message equivocation. Although a ciphertext may be infeasible to break, it is also likely to be unequivocal: that is, it may be impossible to make it appear as the encryption of two different messages. In some sense, this is intuitively unavoidable, since a receiver must be able to decide on a cleartext interpretation. But therein lies the central problem:

> **Equivocation Paradox.** If a ciphertext can be decrypted to more than one cleartext, then a receiver cannot be sure what message it received. If a ciphertext cannot be decrypted to more than one cleartext, then a simulator cannot demonstrate that it is secure.

Advances: Past and Present. Beaver and Haber presented a direct and efficient cryptosystem that requires each party to erase certain internal records [BH92]. Without access to these records, a later attacker will not find sufficient information to determine that a simulated ciphertext is different from an actual ciphertext. A similar construction has been reported to have been developed by Feldman (see [CFGN96]).

Although it is good security practice to erase keys as soon as possible, it is certainly preferable to avoid basing security on such demands. Canetti, Feige, Goldreich, and Naor broke through this barrier with an ingenious method that requires no erasing [CFGN96].

Neither of these methods is fully satisfactory: one requires erasing and careful attention regarding automatic backups, while the other is complex and expensive in both its design and its verification.

This paper capitalizes on important concepts from each and extends them to broader and simpler techniques, achieving quasi-practical (as opposed to carefully-managed or merely theoretic) performance.

Using an unusual twist on Diffie-Hellman key exchange, we present a novel method for adaptively-secure key exchange:

Theorem 1. *There exists a non-erasing implementation of secret key exchange, using expected $O(1)$ invocations of Diffie-Hellman key exchange per bit, that is secure against adaptive 2-adversaries, if the Diffie-Hellman Assumption holds.*

In comparison, the protocol of [BH92] is more efficient but requires erasing, while the protocol of [CFGN96] achieved the same goal already but with $\Omega(k)$ invocations per key bit of a similar underlying primitive, where k is a security parameter.

We employ DH-based solutions for the purpose of exposition. These results generalize naturally to factoring, discrete logarithm, and, if the cost of involving network computations is permissible, to any one-way trapdoor permutation (as in [CFGN96]).

These results also provide the most efficient open-channel replacements available for the private channels used in the information-theoretically secure multiparty computation protocols of Ben-Or, Goldwasser and Wigderson [BGW88] and Chaum, Crépeau, and Damgard [CCD88].

Notably, the proof of security against adaptive attacks is radically simplified, as well. This is particularly important in light of the apparent increase in complexity (and hence risk) of extending formal analysis from static to adaptive scenarios. That is, even though one might expect to encounter more complicated analyses and therefore enjoy less confidence in the results, we demonstrate that this is unnecessary.

Further Remarks: Commitment and Deniability. We have avoided the term "non-committing" [CFGN96] because of certain ambiguities. In particular, all of the above results are "committing" in the sense that an honest sender cannot later pretend that an alternate message was sent. That is, these cryptosystems are non-committing for the *simulator*, and they are non-committing in that they do not immediately serve as a bit commitment scheme as a typical cryptosystem does. These caveats said, our cryptosystem shares the same such properties as [CFGN96].

The protocol presented here is committing yet "equivocable": real ciphertexts are unequivocal and bind an honest sender to the cleartext, but they can be made to appear equivocal using specially-crafted facsimiles.

In separate work, the methodology presented here has been extended to be *non-committal* in a different sense, *viz* an honest sender and receiver can convince outside inspectors that an arbitrary alternative cleartext was sent [B96], enabling them thus to deny that a particular cleartext was sent. The solution presented in [B96] uses a more complicated and less efficient mechanism, and it obscures the primary solution presented here. Moreover, this paper presents a full proof of security.

2 Formalities

Attacks: Static or Adaptive. An adversary is a probabilistic poly-time TM (PPTM) that issues two sorts of messages: "*corrupt* i," "*send* m from i to j." It receives two sorts of responses: "*view of* i," "*receive* m from j to i."

A **static t-adversary** is an adversary who issues up to t *corrupt* requests before the protocol starts. An **adaptive t-adversary** may issue up to t such requests at any time.

Encryption. The **specification protocol for secure channels** is a two-party protocol consisting of $(\hat{A}, \hat{B})$, in which $\hat{A}$ inputs a bit m which is transferred securely to $\hat{B}$. An eavesdropper knows only that a bit was sent, or that one or the other party decided to abort.

To simplify analysis, we consider an implementation network that provides authenticated, service-undeniable, point-to-point connections. Our protocols can otherwise be extended, though such strengthenings are uninstructive, here. The traffic over the lines is public, of course.

Simulation-based security. In the adaptive case, there is a single interface/simulator, $\mathcal{I}$, who receives requests from and delivers responses to the attacker, $\mathcal{A}$, creating an environment for $\mathcal{A}$ as though $\mathcal{A}$ were attacking a given implementation. $\mathcal{I}$ is itself an attacker acting within the specification protocol, which is run with players $\hat{i}$ following the specification's programs on inputs x_i. When $\mathcal{A}$ corrupts player i, $\mathcal{I}$ issues a corruption request and is given $\hat{i}$'s information.[3] $\mathcal{I}$ responds to $\mathcal{A}$ with a facsimile of the "*view of* i" response that $\mathcal{A}$ expects. $\mathcal{I}$ receives all of $\mathcal{A}$'s "*send* m" requests and provides $\mathcal{A}$ with facsimiles of "*receive* m" responses. Finally, $\mathcal{A}$ (or $\mathcal{I}$ on $\mathcal{A}$'s behalf) writes its output, $y_{\mathcal{A}}$.

In the case of secure channels, let $\mathcal{A}$, with auxiliary input $x_{\mathcal{A}}$, attack a given implementation in which Alice holds input m. The execution induces a distribution $(A(m), B, \mathcal{A}(x_{\mathcal{A}}))$ on output triples, $(y_A, y_B, y_{\mathcal{A}})$.

Let $\mathcal{I}(\mathcal{A}(x_{\mathcal{A}}))$ attack the specification (described above). The execution induces a distribution $(\hat{A}(b), \hat{B}, \mathcal{I}(\mathcal{A}(x_{\mathcal{A}})))$ on output triples, $(y_{\hat{A}}, y_{\hat{B}}, y_{\mathcal{I}})$.

An extra, "security" parameter k may be considered. This provides a sequence of distributions on output triples in each scenario. Let $\approx$ denote *computational indistinguishability*, a notion whose formal definition is omitted for reasons of space (*cf.* [GMR89]).

An **encryption scheme secure against adaptive t-adversaries** is a (two-party) protocol such that, for any adaptive t-adversary $\mathcal{A}$, there is a PPTM simulator $\mathcal{I}$ such that for any m, $(A(m), B, \mathcal{A}(x_{\mathcal{A}})) \approx (\hat{A}(m), \hat{B}, \mathcal{I}(\mathcal{A}(x_{\mathcal{A}})))$.[4]

Notation. Let $\$(S)$ denote the uniformly random distribution over finite set S. For a prime p, let $\mathbf{Z}_p^* = \{1, 2, \ldots, p-1\}$ and $\mathbf{Z}_{p-1} = \{0, 1, 2, \ldots, p-2\}$.

Assumptions. Let $p - 1 = 2p'$, where p and p' are prime. Let $\hat{g}$ be a generator of $\mathbf{Z}_p^*$, and define $g = \hat{g}^2 \bmod p$. Then g generates a subgroup $\langle g \rangle$ (of quadratic residues mod p). Define the **Diffie-Hellman distribution** D_p as the triple of random variables (A, B, C) obtained through

$$a \leftarrow \$(\mathbf{Z}_{p-1}), b \leftarrow \$(\mathbf{Z}_{p-1}), A \leftarrow g^a \bmod p, B \leftarrow g^b \bmod p, C \leftarrow g^{ab} \bmod p.$$

The **Decision Diffie-Hellman Assumption** (DDHA) can be described as follows:

> (DDHA) Let p be a prime and g a subgroup generator selected as described above. Then D_p is computationally indistinguishable from $(\$(\langle g \rangle), \$(\langle g \rangle), \$(\langle g \rangle))$.

Note that without the precaution of moving to a subgroup, typical Diffie-Hellman triples can be distinguished from three random elements. The quadratic residuosity of g^{ab} can be deduced from that of g^a and g^b, hence a random element would be distinguishable from g^{ab}.

---

[3] $\hat{i}$ is a player in the specification protocol and is unaware of messages being passed in a given implementation. In particular, $\hat{i}$ knows only its input x_i and the messages it sends and receives over channels supported in the specification.

[4] A multiparty implementation is also possible; the formalities are similar.

3 Adaptively Secure Key Exchange

The novel idea (and surprising twist) behind our protocols is two-fold: although Alice and Bob engage in a classical, statically-secure key exchange protocol, they (1) sometimes garble their strings and (2) always reveal the secret key!

More specifically, Alice and Bob perform two parallel DH exchanges (indexed 0 and 1, say). They independently choose to garble computations and strings on precisely one index. Bob responds not only with (possibly garbled) g^b but with the (possibly garbled) key in each exchange.

If they chose identical indices to garble, then the key will be recognizably correct in the ungarbled instance. This provides them with a common secret bit: the index of the ungarbled instance. Of course, if they chose opposite indices, then they detect garbage on both instances and try again (sequentially or in parallel).

An eavesdropper, Eve, learns only whether Alice and Bob made the same guess in a given attempt, but not what that guess was. It remains to determine whether the details ensure three constraints: (1) Eve cannot tell the difference between garbage and a valid exchange; (2) Alice and Bob can agree on which index is garbled; and critically, (3) a simulator can construct conversations that can later be made consistent with (valid,garble) or (garble,valid) as needed.

3.1 Creating Garbage

The "garbling" of the Diffie-Hellman protocol occurs in one of two ways. Instead of choosing an exponent e and computing $r = g^e$, a player can choose $r \in \langle g \rangle$ directly without knowing its discrete logarithm. Naturally, that player will be unable to calculate or verify the final DH key, g^{ab}, but this is unimportant to Alice and Bob. (It is *extremely* important to the proof of security, however!) Second, a player can garble g^{ab} by likewise choosing a uniformly random residue whose discrete logarithm is unknown.

Fig. 1 describes the full protocol for a single attempt to transmit a one-bit message, m. Alice and Bob each conclude either `fail` or `succeed`:m. Clearly, they can trivially establish a one-time pad bit instead by using $m = 0$.

Note that we have no need of implicit zero-knowledge proofs of behavior or knowledge, or even simple verification beyond parsability: a corrupt Alice or Bob is fully permitted to learn all available logarithms and to force the agreed-upon bit to be anything they like.

If Alice's choice, c, matches Bob's choice, d, then the relevant variables describe a normal execution of Diffie-Hellman key exchange. In particular, the Diffie-Hellman key would be $x_c^b = y_c^a = z_c$. Thus, Alice can simply check whether $y_c^a = z_c$.[5] If they are equal, Alice knows $c = d$ and uses c to mask m.

[5] We are concerned only with sharing a single, equivocal random bit, thus revealing the value z_d is not an issue. Alternatively, the key can be hashed or used to encrypt a known or redundant message in order to detect whether the chosen index "makes sense" or whether it is garbled.

Send-Bit-Attempt(m)

0.	Public:	prime p, subgroup generator g	3.1.	A:	if $y_c^a = z_d$, then $s \leftarrow 0$
1.1.	A:	$c \leftarrow \$(\{0,1\})$, $a \leftarrow \$(\mathbf{Z}_{p-1})$,			else $s \leftarrow 1$
		$x_c \leftarrow g^a \bmod p$, $x_{1-c} \leftarrow \$(\langle g \rangle)$			if $s = 0$, then $f \leftarrow m \oplus c$
1.2.	A→B:	x_0, x_1			else $f \leftarrow 0$
2.1.	B:	$d \leftarrow \$(\{0,1\})$, $b \leftarrow \$(\mathbf{Z}_{p-1})$,	3.2.	A→B:	(s, f)
		$y_d \leftarrow g^b \bmod p$, $y_{1-d} \leftarrow \$(\langle g \rangle)$,	4.1.	B:	if $s = 0$, then
		$z_d \leftarrow x_d^b \bmod p$, $z_{1-d} \leftarrow \$(\langle g \rangle)$			$m \leftarrow f \oplus d$
2.2.	B→A:	y_0, y_1, z_0, z_1			conclude succeed:m
					else conclude `fail`

Fig. 1. Three-pass attempt to transmit one bit, m.

If, however, their choices differ, then Bob has chosen both y_c and z_c as random residues, thus with high probability, $y_c^b \neq z_c$. Alice therefore informs Bob of failure.

It may seem possible to utilize even the failed attempts, since Alice could nevertheless calculate $d = 1 - c$ and use d as a one-time pad bit. This unfortunately disables equivocation by the simulator.

3.2 Three Passes or Four?

To establish a 1-bit shared secret key (or exchange a 1-bit message) with high probability, it suffices to use k parallel attempts. To establish a k-bit shared secret key, $3k$ parallel attempts clearly suffice.

(Note that, unlike the case of parallel zero-knowledge, we do not face the issue of mutually antagonistic parties attempting to withhold information from one another. Even if one of the parties is malicious, there are no challenges which a simulator needs to overcome, despite lacking a secret proof (or message): here, the simulator would have *all* knowledge, including any desired m, and it trivially simulates an honest party.)

Sending a k-bit message requires a touch more thought, however. Clearly, a fourth pass will suffice. The fourth pass could be avoided by using k invocations of 1-message-bit exchange, resulting in $O(k^2)$ attempts. Instead, applying the linear codes of Sipser and Spielman [SiSp94], a k-bit message can be transmitted in 3 passes using only $O(k)$ attempts.

4 Proof of Security

The motivation for our counterintuitive disposal of DH is to enable a simulator to produce a fake conversation that can be explained as representing either a 0 or a 1, without yet knowing which explanation will be required. Normally, this is impossible, since a standard key exchange will uniquely define a secret. (Indeed, even this protocol commits an honest Alice and Bob to the secret bit.)

Interface-No-Corruption

0. $\mathcal{I} \to \mathcal{A}$: prime p, subgroup generator g

1.1. Internal: $a_0 \leftarrow \$(\mathbb{Z}_{p-1})$, $a_1 \leftarrow \$(\mathbb{Z}_{p-1})$
$x_0 \leftarrow g^{a_0} \bmod p$, $x_1 \leftarrow g^{a_1} \bmod p$

1.2. $\mathcal{I} \to \mathcal{A}$: "A→B: x_0, x_1"

2.1. Internal: $s \leftarrow \$(\{0,1\})$
$b_0 \leftarrow \$(\{0,1\})$, $b_1 \leftarrow \$(\{0,1\})$
$y_0 \leftarrow g^{b_0} \bmod p$, $y_1 \leftarrow g^{b_1} \bmod p$
if $s = 0$ then
$z_0 \leftarrow g^{a_0 b_0} \bmod p$,
$z_1 \leftarrow g^{a_1 b_1} \bmod p$
(cont.)

(2.1 cont.) else
$c \leftarrow \$(\{0,1\})$,
$a \leftarrow a_c$,
$d \leftarrow 1 - c$,
$b \leftarrow b_d$,
$z_d \leftarrow y_d^{b_d} \bmod p$,
$z_{1-d} \leftarrow \$(\langle g \rangle)$

2.2. $\mathcal{I} \to \mathcal{A}$: "B→A: y_0, y_1, z_0, z_1"

3.1. Internal: if $s = 0$ then $f \leftarrow \$(\{0,1\})$
else $f \leftarrow 0$

3.2. $\mathcal{I} \to \mathcal{A}$: "A→B: (s, f)"

Fig. 2. Single-attempt interface/simulator, without corruption.

The simulator uses clean garbage, however. It constructs a conversation containing two *valid* DH exchanges, for which it knows *all* discrete logarithms. It can later pretend that both parties chose index c by witholding the discrete logs for exchange $1 - c$. Even though the resulting conversation is information-theoretically distinct from a real conversation distribution, a poly-bounded judge cannot detect the difference without breaking DDHA.

Our solution expands the intuition implicit in [CFGN96], in which the receiver is helped to avoid learning full information. Here, we arrange for *both* sender and receiver to avoid learning full information.

To prove security against adaptive adversaries, we present an interface that, when attacking an interaction between Alice and Bob over an absolutely secure channel, provides an adversary $\mathcal{A}$ with a fake view that is computationally indistinguishable from a real one. In the past, the great difficulty lay in patching a partially-committed view to accommodate $m_i = 0$ or $m_i = 1$ flexibly, particularly when one or both parties may be corrupted much later on.

For the moment, let us focus on a single execution of `Send-Bit-Attempt`. We first describe the action of the interface $\mathcal{I}$ when $\mathcal{A}$ makes no corruption requests. Unlike Alice and Bob, $\mathcal{I}$ "cheats" by discovering *both* keys. Instead of setting x_{1-c} and y_{1-d} to random values with unknown discrete logs, as an honest Alice or Bob would do, $\mathcal{I}$ knowingly selects their logarithms at random. In doing so, $\mathcal{I}$ retains the ability to "open" x_0 or x_1 consistently with $c = 0$ or $c = 1$. Moreover, $\mathcal{I}$ does not garble z_{1-d}, but improperly uses $x_{1-d}^{\log_g y_{1-d}}$. See Fig. 2 for details.

The key is that *the logarithms of the critical values are never explicitly represented anywhere in the network* – not even as a shared secret. Thus even an adversary who gains *both* Alice's and Bob's *complete, unerased* internal histories cannot calculate the discrete logs or even distinguish the critical residues from random values. (Naturally, a computationally-*unbounded* adversary will be able to distinguish $\mathcal{I}$'s fake view from a real-life view.)

In sum, $\mathcal{I}$ generates three fake messages:

$$(x_0, x_1), (y_0, y_1, z_0, z_1), (s, f)$$

where, if $s = 0$, then $z_i = y_i^{\log_g x_i}$.

Let us call x_{1-d}, y_{1-d}, and z_{1-d} "critical variables." In an execution of the **Send-Message** protocol in which Alice's message is generated honestly, the critical variables are uniformly random and independent of all other variables. The significant and sole difference between an actual transcript and $\mathcal{I}$'s generated version is that in $\mathcal{I}$'s version, when $s = 1$, the critical variables are *not independent* – although they are individually uniformly random. Specifically, $z_{1-d} = x_{1-d}^{\log_g y_{1-d}}$ – which is a relationship that does not hold when honest Alice generates the three values. This relationship is, however, infeasible to detect.

4.1 Corruption Requests

We now turn to how the interface handles corruption requests. There are four cases, depending on when $\mathcal{A}$ makes its first corruption request.

Case 0: $\mathcal{A}$ makes its first corruption request before any messages are sent.
Case 1: $\mathcal{A}$ makes its first corruption request after Alice sends her message, but before Bob sends his message.
Case 2: $\mathcal{A}$ makes its first corruption request after Alice and Bob have each sent one message.
Case 3: $\mathcal{A}$ makes its first corruption request after Alice and Bob have sent all their messages.

When the first corruption request is made, $\mathcal{I}$ will "patch" the views of *both* players, hand over the view of the corrupted player to $\mathcal{A}$, and then assume the role of the uncorrupted player based on the patched view. Note that the message bit m plays no role until Alice's second message to Bob.

Patching fake views. We turn first to how $\mathcal{I}$ patches the fake views. For clarity, the handling of auxiliary inputs (such as histories from previous protocols) is left implicit.

We further subdivide the cases into 0A, 0B, 1A, 1B, 2A, 2B, 3A and 3B, according to whether $\mathcal{A}$ selects Alice or Bob to corrupt first.
Case 0.A: $\mathcal{I}$ obtains $\hat{A}$'s input m from corrupting $\hat{A}$ in the specification protocol and reports it to $\mathcal{A}$.
Case 0.B: Nothing to patch.

Case 1.A: $\mathcal{I}$ obtains $\hat{A}$'s input m from the specification protocol. $\mathcal{I}$ performs $c \leftarrow \$(\{0,1\})$; $a \leftarrow a_d$. $\mathcal{I}$ patches A's view with m, c, a, (x_0, x_1) and reports it to $\mathcal{A}$. $\mathcal{I}$ patches B's view with (x_0, x_1).
Case 1.B: $\mathcal{I}$ patches B's view with (x_0, x_1).

Case 2.A: $\mathcal{I}$ obtains $\hat{A}$'s input m from the specification protocol. $\mathcal{I}$ performs $c \leftarrow \$(\{0,1\})$; $a \leftarrow a_d$; $d \leftarrow \$(\{0,1\})$; $b \leftarrow b_d$. $\mathcal{I}$ patches A's view with

m, c, a, $(x_0, x_1), (y_0, y_1, z_0, z_1)$, and reports it to $\mathcal{A}$. $\mathcal{I}$ patches B's view with $(x_0, x_1), d, b_c, (y_0, y_1, z_0, z_1)$.

Case 2.B: $\mathcal{I}$ performs $c \leftarrow \$(\{0,1\})$; $a \leftarrow a_d$; $d \leftarrow \$(\{0,1\})$; $b \leftarrow b_d$. $\mathcal{I}$ patches A's view with c, a, $(x_0, x_1), (y_0, y_1, z_0, z_1)$. $\mathcal{I}$ patches B's view with $(x_0, x_1), d, b_c, (y_0, y_1, z_0, z_1)$ and reports it to $\mathcal{A}$.

Case 3.A: $\mathcal{I}$ obtains $\hat{A}$'s input m from the specification protocol. At this point, $\mathcal{I}$ has irrevocably decided whether this attempt will succeed ($s = 0$) or fail ($s = 1$).

If $s = 0$, perform the following: $c \leftarrow m \oplus f$; $a \leftarrow a_c$; $d \leftarrow c$; $b \leftarrow b_d$.

If $s = 1$, $\mathcal{I}$ has already chosen values for c, a_c, d, and b_d.

$\mathcal{I}$ patches A's view with c, a, $(x_0, x_1), (y_0, y_1, z_0, z_1), (s, f)$, and reports it to $\mathcal{A}$. $\mathcal{I}$ patches B's view with $(x_0, x_1), d, b_c, (y_0, y_1, z_0, z_1), (s, f)$.

Case 3.B: $\mathcal{I}$ obtains message bit m from the secure channel (after honest $\hat{A}$ sends it to now-corrupt $\hat{B}$). As in case 3.A., $\mathcal{I}$ has irrevocably decided whether this attempt will succeed ($s = 0$) or fail ($s = 1$).

If $s = 0$, perform the following: $c \leftarrow m \oplus f$; $a \leftarrow a_c$; $d \leftarrow c$; $b \leftarrow b_d$.

If $s = 1$, $\mathcal{I}$ has already chosen values for c, a_c, d, and b_d.

$\mathcal{I}$ patches A's view with c, a, $(x_0, x_1), (y_0, y_1, z_0, z_1), (s, f)$. $\mathcal{I}$ patches B's view with $(x_0, x_1), d, b_c, (y_0, y_1, z_0, z_1), (s, f)$, and reports it to $\mathcal{A}$.

Assuming the role of the remaining, honest player. Generally speaking, $\mathcal{I}$ merely runs an internal copy of Alice or Bob, having set Alice's or Bob's state according to the given view. Should $\mathcal{A}$ then corrupt the remaining honest player, $\mathcal{I}$ simply hands over the current view. If the remaining honest player is Bob, this procedure is straightforward, since Bob has no special input.

If the honest player is Alice, however, $\mathcal{I}$ plays Alice's role without knowing m, at least until step 3 of `Send-Bit-Attempt`. Since Alice's computations do not depend on m until then, this presents no problem. Once $\mathcal{I}$ has received $\mathcal{A}$'s message to honest Alice on behalf of corrupt Bob, $\mathcal{I}$ waits for corrupt $\hat{B}$ to receive m along the secure channel. It then resumes its internal simulation of honest Alice.

Deciding what to send. If Alice is corrupted before she sends her second message, or equivalently, before $\hat{A}$ has sent the message m, then $\mathcal{I}$ must decide what to send on the secure channel on behalf of the now-corrupt $\hat{A}$. In this case, $\mathcal{I}$ did not yet commit a fake second message to $\mathcal{A}$; that is, $\mathcal{I}$ obtains Alice's message (s, f) from $\mathcal{A}$. $\mathcal{I}$ then continues to run the honest, internal copy of Bob, deriving Bob's effective result, either `fail` or `succeed:`m. In the former case, $\mathcal{I}$ does not send a bit (or append a bit to a longer secure message, in the context of a k-bit protocol). In the latter case, $\mathcal{I}$ sends the bit (resp., appends the bit m to the secure message from corrupt $\hat{A}$).

Syntactic errors. The response of an honest player to a syntactic error (messages that cannot be parsed, *etc.*) is to abort the protocol (resp., place the ideal secure channel in a publicly aborted state). The necessary refinements to the preceding discussion are obvious, tedious, and omitted.

4.2 Reduction to DDHA

By inspection, the distribution that $\mathcal{I}$ hands to $\mathcal{A}$ is identical to that obtained in an actual execution, *except for* the lack of independence among the critical variables, when transmission is successful. It remains to show that this lack of independence is unnoticeable to $\mathcal{A}$. In particular, if it were detectable (to polynomial-bounded observers), then DDHA would fail.

Note that when Alice or Bob is corrupted before Alice sends her second message, the critical variables raise no concerns: they are either properly independent (because they follow honest Alice's program) or generated by an $\mathcal{A}$-controlled Alice. In particular, the results obtained when Alice or Bob is corrupt by the time Alice sends her second message are identical whether obtained through the protocol or through $\mathcal{I}$. The interesting case occurs when neither Alice nor Bob is corrupt at the time Alice sends her message – that is to say, at the time $\mathcal{I}$ supplies Alice's second message to $\mathcal{A}$.

Suppose that the implementation were insecure, namely that there were a poly-time machine D that distinguishes $(A, B, \mathcal{A})$ from $(\hat{A}, \hat{B}, \mathcal{I}(\mathcal{A}))$ with success probability $1/2 + k^{-c}$, for some fixed $m \in \{0, 1\}$, $c > 0$ and infinitely many k. Call any such k "vulnerable."

We describe an algorithm, **Break**, that violates the DDHA. The input to **Break** consists of three values (in addition to g and p):

$$g^{\alpha}, g^{\beta}, \gamma.$$

Let distribution δ_0 generate these as follows: $\alpha \leftarrow \$(\mathbf{Z}_{p-1})$; $\beta \leftarrow \$(\mathbf{Z}_{p-1})$; $\gamma \leftarrow g^{\alpha\beta}$. In contrast, distribution δ_1 applies the following: $\alpha \leftarrow \$(\mathbf{Z}_{p-1})$; $\beta \leftarrow \$(\mathbf{Z}_{p-1})$; $\gamma \leftarrow \$(\langle g \rangle)$. Our goal is to distinguish δ_0 from δ_1, thereby violating the DDHA.

Observe particularly that in our earlier construction, $\mathcal{I}$ needs to know all four values a_0, a_1, b_0, and b_1, because it does not know what m will be, but it may have to equivocate later on. The converse is also true: *knowing the result* **fail** *or* **succeed***:m, one does not need to know all four logarithms in order to duplicate the behavior of* $\mathcal{I}$. The **Break** routine takes advantage of this fact.

Intuitively, $\mathcal{I}$ must stick with the s and f it selected without knowing m, and (when $s = 0$) it later adapts d to its discovery of m, so that $d = m \oplus f$. On the other hand, **Break** can select d and s, and when $s = 0$, knowing m already, it *calculates* $f = m \oplus d$ instead of choosing f independently at random.

In more detail, **Break** runs an internal execution of the specification protocol, permitting a built-in interface $\mathcal{I}_{brk}$ to interact with $\mathcal{A}$. **Break** supplies its input (sampled from δ_0 or δ_1) to $\mathcal{I}_{brk}$. **Break** knows m and can therefore operate internal copies of $\hat{A}$ and $\hat{B}$ as well. Depending on the distribution given to **Break**, the final distribution is identical to either (0) an attack by $\mathcal{A}$ assisted by $\mathcal{I}$ against the specification protocol, or (1) an attack by $\mathcal{A}$ on the **Send-Message** protocol. Once **Break** has obtained the final results, it passes them to an internal copy of distinguisher D and simply reports whatever D reports.

The built-in interface $\mathcal{I}_{brk}$ follows the general outline of $\mathcal{I}$'s program, except that $\mathcal{I}_{brk}$ commits to d and s *without* knowing the discrete logarithms of all the

$\mathcal{I}_{brk}(g^\alpha, g^\beta, \gamma)$

0. Given: prime p, subgroup generator g

1.1. Internal: $c \leftarrow \$(\{0,1\})$,
$a_c \leftarrow \$(\mathbf{Z}_{p-1})$,
$x_c \leftarrow g^{a_c} \bmod p$,
$x_{1-c} \leftarrow g^{\alpha} \bmod p$

1.2. $\mathcal{I} \rightarrow \mathcal{A}$: "A→B: x_0, x_1"

2.1. Internal: $s \leftarrow \$(\{0,1\})$
if $s = 0$ then
$d \leftarrow c$,
$b_d \leftarrow \$(\mathbf{Z}_{p-1})$,
$y_d \leftarrow g^{b_d} \bmod p$,
$z_d \leftarrow g^{a_d b_d} \bmod p$,
$y_{1-d} \leftarrow g^{\beta} \bmod p$,
$z_{1-d} \leftarrow \gamma \bmod p$

(2.1 cont.) else
$d \leftarrow 1 - c$,
$b_d \leftarrow \$(\mathbf{Z}_{p-1})$,
$y_d \leftarrow g^{b_d} \bmod p$,
$z_d \leftarrow g^{\alpha b_d} \bmod p$,
$y_{1-d} \leftarrow \$(\langle g \rangle)$,
$z_{1-d} \leftarrow \$(\langle g \rangle)$

2.2. $\mathcal{I} \rightarrow \mathcal{A}$: "B→A: y_0, y_1, z_0, z_1"

3.1. Internal: if $s = 0$ then
$f \leftarrow c \oplus m$,
else
$f \leftarrow 0$

3.2. $\mathcal{I} \rightarrow \mathcal{A}$: "A→B: (s, f)"

Fig. 3. Behavior of $\mathcal{I}_{brk}$, used as a subroutine to violate DDHA.

values. It does not get caught trying to equivocate, because it already knows m. Fig. 3 describes the details.

Fig. 4 illustrates the results induced by $\mathcal{I}_{brk}$. If $\gamma = g^{\alpha\beta}$, these tables correspond to the results of $\mathcal{I}$. If γ is chosen independently at random, these tables correspond to the results of executing `Send-Bit-Attempt`.

The following two observations are straightforward. If $(g^\alpha, g^\beta, \gamma) \leftarrow \delta_0$, then

$$(\hat{A}(m,k), \hat{B}(k), \mathcal{I}_{brk}(g^\alpha, g^\beta, \gamma; \mathcal{A}(k), k, m)) = (\hat{A}(m,k), \hat{B}(k), \mathcal{I}(\mathcal{A}(k), k)).$$

If $(g^\alpha, g^\beta, \gamma) \leftarrow \delta_1$, then

$$(\hat{A}(m,k), \hat{B}(k), \mathcal{I}_{brk}(g^\alpha, g^\beta, \gamma; \mathcal{A}(k), k, m)) = (A(m,k), B(k), \mathcal{A}(k)).$$

By the construction of `Break`,

$$\texttt{Break}(\delta_0) = D((\hat{A}(m,k), \hat{B}(k), \mathcal{I}(\mathcal{A}(k), k)))$$
$$\texttt{Break}(\delta_1) = D((A(m,k), B(k), \mathcal{A}(k))).$$

Thus, $|\texttt{Break}(\delta_0) - \texttt{Break}(\delta_1)| \geq 1/2 + k^{-c}$ at all vulnerable k, contradicting DDHA as desired.

4.3 Full Key Exchange

When establishing a k-bit key K using $3k$ parallel attempts, we randomly select $i \leftarrow \$(\{1, \ldots, 3k\})$ and use the above `Break` routine in the i^{th} parallel iteration. We generate the portions of Alice's second message in the locations $1..(i-1)$ using $\mathcal{I}$. We generate the portions of Alice's second message in the locations $(i+1)..3k$ according to `Send-Bit-Attempt`.

Through standard arguments, we conclude that our algorithm distinguishes δ_0 from δ_1 with advantage $1/2 + (k^{-c}/3k)$ at all vulnerable k. This suffices to contradict the DDHA, extending the proof to k-bit exchanges.

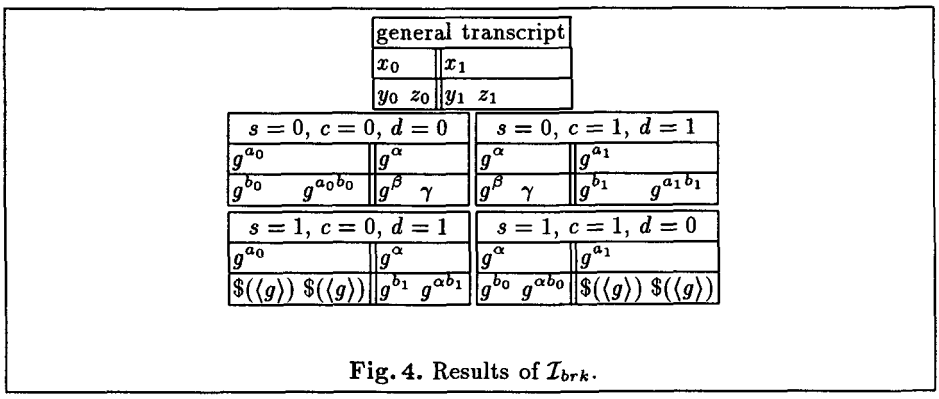

Fig. 4. Results of $\mathcal{I}_{brk}$.

5 Concluding Remarks

Although it appears that Alice could calculate and use d in all cases (a mismatch informs her that $d = 1 - c$), this simple optimization fails for technical reasons. It is not hard to show that $\mathcal{I}$'s fake encryptions must contain two DH triples as before, but in this case they are easily unveiled by detecting whether $z_{1-d} = y_{1-d}^{a_c}$.

Generalizations. To use other intractability assumptions, such as RSA or factoring, a suitable key-exchange construction suffices. In particular, the dense secure public-key cryptosystems of DeSantis and Persiano are appropriate [DP92].

When third parties are available, one-way trapdoor permutations suffice. Unlike [CFGN96], *both* Alice and Bob obtain one of two composed trapdoors; instead of applying large-scale permutations of multiple encryptions, they then detect a matched choice in the simple manner suggested here.

Committal and Equivocability. We identify two key properties useful for analyzing and designing encryption systems robust enough for plug-and-play usage:

1. *Equivocability:* the ability to make convincing fake ciphertexts that are equivocal, even if the real ciphertexts are not.
2. *Selective Ignorance:* arranging for a party to avoid learning full information.

The results of [BH92] apply the first approach, destroying sufficient information to allow ciphertexts to be made consistent with different cleartexts. Note that the *real* ciphertexts are ultimately equivocal; the cryptosystem cannot be used indirectly as a committal.

The methods in [CFGN96] identified and applied the second approach as well. In their protocol, a clever if costly interaction enables the sender to obtain partial information from the receiver and/or third parties. The lack of full information permits a simulator to create equivocal fake ciphertexts, even though real ciphertexts are unequivocal. Certain roots can be seen in DeSantis' and Persiano's work on non-interactive zero-knowledge proofs [DP92].

The current work applies this principle one step further, arranging for *both* the sender and receiver to avoid learning certain information. The result is far greater simplicity and efficiency.

As is the case with [CFGN96] but not [BH92], the approach presented here binds the parties (not the simulator) to the cleartext message. Through a slightly more expensive and involved generalization, the parties can enjoy the ability to pretend to outside inspectors that an arbitrary alternative cleartext was sent [B96], enabling them to deny having sent their actual messages.

References

[B91] D. Beaver. "Foundations of Secure Interactive Computing." *Advances in Cryptology – Crypto '91 Proceedings,* Springer–Verlag LNCS 576, 1992, 377–391.

[B95] D. Beaver. "Adaptive Zero Knowledge and Computational Equivocation." *Proceedings of the* 28^{th} *STOC,* ACM, 1996, 629–638.

[B96] D. Beaver. "Plausible Deniability." *Advances in Cryptology – Pragocrypt '96 Proceedings,* CTU Publishing House, 1996, 272–288.

[BH92] D. Beaver, S. Haber. "Cryptographic Protocols Provably Secure Against Dynamic Adversaries." *Advances in Cryptology – Eurocrypt '92 Proceedings,* Springer–Verlag LNCS 658, 1993, 307–323.

[BGW88] M. Ben-Or, S. Goldwasser, A. Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation." *Proceedings of the* 20^{th} *STOC,* ACM, 1988, 1–10.

[CFGN96] R. Canetti, U. Feige, O. Goldreich, M. Naor. "Adaptively Secure Multiparty Computation." *Proceedings of the* 28^{th} *STOC,* ACM, 1996, 639–648.

[CCD88] D. Chaum, C. Crépeau, I. Damgård. "Multiparty Unconditionally Secure Protocols." *Proceedings of the* 20^{th} *STOC,* ACM, 1988, 11–19.

[DP92] A. DeSantis, G. Persiano. "Zero-Knowledge Proofs of Knowledge Without Interaction." *Proceedings of the* 33^{rd} *FOCS,* IEEE, 1992, 427–436.

[DH76] W. Diffie, M. Hellman. "New Directions in Cryptography." *IEEE Transactions on Information Theory* **IT-22**, November 1976, 644–654.

[GM84] S. Goldwasser, S. Micali. "Probabilistic Encryption." *J. Comput. Systems Sci.* **28**, 1984, 270–299.

[GMR89] S. Goldwasser, S. Micali, C. Rackoff. "The Knowledge Complexity of Interactive Proof Systems." *SIAM J. on Computing* **18**:1, 1989, 186–208.

[GMW86] O. Goldreich, S. Micali, A. Wigderson. "Proofs that Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design." *Proceedings of the* 27^{th} *FOCS,* IEEE, 1986, 174–187.

[MR91] S. Micali, P. Rogaway. "Secure Computation." *Advances in Cryptology – Crypto '91 Proceedings,* Springer–Verlag LNCS 576, 1992, 392–404.

[RSA78] R. Rivest, A. Shamir, L. Adleman. "A Method for Obtaining Digital Signatures and Public Key Cryptosystems." *Communications of the ACM* **21**:2, 1978, 120–126.

[SiSp94] M. Sipser, D. Spielman. "Expander Codes." *Proceedings of the* 35^{th} *FOCS,* IEEE, 1994, 566–576.