Proofs for Two-Server Password Authentication

Michael Szydlo and Burton Kaliski

RSA Laboratories
Bedford, MA 01730, USA
E-mail: {mszydlo,bkaliski}@rsasecurity.com

Abstract. Traditional password-based authentication and key-exchange protocols suffer from the simple fact that a single server stores the sensitive user password. In practice, when such a server is compromised, a large number of user passwords, (usually password hashes) are exposed at once. A natural solution involves splitting password between two or more servers. This work formally models the basic security requirement for two-server password authentication protocols, and in this framework provides concrete security proofs for two protocols. The first protocol considered [7] appeared at USENIX'03, but contained no security proof. For this protocol, we provide a concrete reduction to the computational Diffie-Hellman problem in the random oracle model. Next we present a second protocol, based on the same hard problem, but which is simpler, and has an easier, tighter reduction proof.

Key words: password authentication, secret sharing, concrete security reduction

1 Introduction

Passwords remains the most widespread method of user authentication to date, despite their inherent weaknesses. For example, user passwords, or password hashes are often stored in a server database, and the user authenticates by sending the password back using a server-side SSL authenticated channel. Of course, all password systems permit an attacker to make some number of guesses before the server locks the account down. However, a much more serious vulnerability exists: in case of a server compromise, an attacker may obtain *all* user passwords, or password hashes in the database at once.

Strengthening Passwords: The convenience of user-chosen password authentication protocols has caused them to be widely deployed. Among the weaker protocols one finds passwords sent in the clear, reusable low entropy PINs, and hash based challenge response techniques. A commonly used, and better, approach is to send a password or password hash over a server-side SSL- authenticated connection. Conceptually, these approaches still suffer from the fact that a user may be tricked into revealing his password to a server who does not know it. Starting with *Encrypted Key Exchange*[2] of Bellovin and Merritt, the benefits of a zero-knowledge based approach were realized. The goal of such protocols is to provide an authentication procedure which does not reveal a user password to any party who does not already have it. This line of research continued in several directions [8, 14, 4, 12], and represent a significant improvement in client-server protocols.

Multiple Server Use: Despite the improvements described above, single server password based authentication protocols do not protect from server compromise in a satisfactory way. Typically, an attacker who breaches a server will be able to obtain a very large number of user passwords, perhaps after running a dictionary attack (salt merely slows this). The natural approach to addressing this vulnerability is the use of multiple servers. In such schemes, the capability of verifying a password is split among two or more machines, and more than a certain threshold number of servers need to collude to recover the password. Starting with the work of Ford and Kaliski[6], various zero-knowledge multiple server password protocols have been proposed [9, 13, 7]. Multi-server protocols should provide basic username-password authentication to the collection of servers, without using special hardware or long-term client side key storage. Even for low-entropy passwords, an attacker should not be able to improve upon the naive guessing strategy without corrupting a threshold number of servers. On the other hand, these protocols do not pretend to have unrealistic goals of preventing denial of service or protecting user passwords in the case of client compromise.

Our focus, the two-server approach, is appealing for several reasons. With just two servers, the largest risk of wholesale password theft is greatly diminished. Also, deploying a large number of independently run servers appears logistically challenging, whereas the addition of a second server may be feasible in practice.

Provable Security: Increasingly, it has been realized that the proposal of a cryptographic scheme is only as valuable as its accompanying provable security analysis. The security proof techniques based on complexity theoretic foundations, including the copious general results on secure multi-party computation and threshold cryptography, provide tools for analyzing the kinds of protocols we are interested in. Typically, this framework is used to present asymptotic security definitions and security proofs[1]. However, for a protocol which is to be deployed, a *concrete* security analysis is required.

Our Contributions: In this work, we describe an appropriate framework for analyzing the concrete security of two-server password based authentication schemes. For a (random oracle) variant of [7], we provide a concrete reduction to the computational Diffie-Hellman problem. We also present a second protocol, based on the same hard problem, whose security proof is tighter, and simpler. All of our the security proofs are presented as explicit reduction algorithms which relate the difficulty of two computational problems. This approach allows for a more transparent concrete security analysis. Given that difficult security proofs, are sometimes left unread, we hope that our explicit approach is helpful.

[1] The somewhat misleading identification of the term “polynomial time” with “efficient” is due to the notion’s stability among different models of computation.

1.1 Organization

The rest of this paper is organized as follows. In Section 2, we discuss the structure and desired security properties of a two-server password authentication protocol. In Section 3, we describe a general framework for concrete security reductions. In Section 4, we recall the protocol from [7], and present a new protocol. In Section 5, we define the concrete adversarial experiments appropriate for our two party authentication protocols. In Section 6 and 7 we provide the explicit reduction algorithm and state the concrete security result for these schemes. We summarize the results and conclude in Section 9. In the appendix, we discuss the unmodified scheme of [7], and the use of a Decisional Diffe-Hellman oracle.

2 Communication Framework and Desired Security

A two server authentication protocol involves a *Client* and two servers. Following [7], the two servers will be denoted *Blue* and *Red.* During an enrollment phase, the user chooses a password, which is processed by the client to produce registration messages for each server. Later, when a claimant enters a password, the client prepares and sends authentication messages to each server. After the two servers complete a verification protocol, the claimant is notified of the result by one or both servers.

To model a scenario in which the identities of the Blue and Red servers are easily ascertained, we assume that all parties employ a *secure channel* to Blue and Red. In practice, this can be realized with SSL. Architecturally, it may be desirable for the client to communicate with a single server, and this is easily accomplished by treating one server (say Blue) as a router.

The reader will easily verify that the protocols we describe are *complete*; a claimant with correct password will always authenticate correctly. More difficult is to show the soundness property: that an adversary can not do much better than password guessing.

Password Privacy: In this paper, we are interested in measuring if the two-party password protocols optimally protects the sensitive password data in the event that one server is compromised, and that compromising one server does not help an adversary authenticate to the other server. An experiment to test this should be designed so that an adversary

1. Tries to authenticate as a user who has previously enrolled.
2. May compromise one server, gaining the ability to impersonate messages.
3. May pose as the user and interact in some number of rounds, (denoted T).
4. May prompt the actual user to authenticate with the correct password.
5. Is allowed some bounded number of random oracle calls, (denoted Q)[2].
6. Is compared with an ideal-world adversary, allowed T password guesses.

[2] Artificially counting random oracle calls this way is a feature of random oracle security proofs. When arguing that the security carries over to protocols using a hash function, Q is usually set to be proportional to the adversary's running time.

To simplify the formal experiments which follow, we make an additional assumption on the protocol. Namely we assume the two servers employ a session management technique which precludes simultaneous authentication attempts for the same username, and also eliminate attacks which confuse messages corresponding to distinct usernames[3]. This means that the adversary will not gain any advantage by interleaving messages among concurrent sessions.

Limitations of the Model: For concreteness, we set the adversarial goal to be persuading the non-corrupted server to authenticate the adversary as user *username*. It is straightforward to alter the exact experiments described below for the goal of correctly guessing the password. This can be a more natural goal, for instance, when one server (Blue) is granting access to some service, and another (Red) is present to elliminate a single point of password compromise. Then, the natural goal of an adversary compromising Blue is to learn user passwords.

The adversarial capabilities described above do not measure the potential advantage an adversary might gain from *keying error.* Since the adversary is only allowed to activate the client on the *correct* password, the model does not capture the potential advantage for an adversary who observes a client launching the authentication sequence with a incorrect but related password. Although it is clumsy to model, it is conceivable that an adversary might benefit from this.

3 Formal Security Model

3.1 Adversarial Experiment Background

Parties and Experiments: All cryptographic parties are modeled as known stateful, probabalistic algorithms, whose inputs and outputs are interpreted as messages. The adversary, denoted *Adv*, is modeled by an arbitrary, unspecified algorithm. *Experiments*, or *hard problems* are algorithms which call one or more parties (black box) and output a bit $\{0, 1\}$, and are used to describe joint properties of the parties. For each adversary *Adv*, we denote the *running-time* of *Adv* in experiment *Exp* by $Time_{Exp}(Adv)$, and the *success* by $Succ_{Exp}(Adv) = Prob[Exp^{Adv}() = 1]$.

Adversarial Capabilities: A *security property* of a protocol is defined in terms of an experiment which specifies both the *adversarial capabilities* (limiting the number and order of messages sent), and the *adversarial objective*, or success criteria. These experiments are designed to measure the ability of an attacker to disrupt normal protocol flow, or to learn a secret.

Concrete Reduction: A *concrete reduction* from hard problem Exp_{Re} to Exp_{Hp} is a black box conversion of adversary Adv_{Re} for the first to an adversary Adv_{Hp} for the second. More specifically, it consists of:

1. An algorithm *Reduce*, defining $Adv_{Hp} = Reduce^{Adv_{Re}}$.
2. A formula for $Succ_{Exp_{Hp}}(Adv_{Hp})$ in terms of $Succ_{Exp_{Re}}(Adv_{Re})$.
3. A formula for $Time_{Exp_{Hp}}(Adv_{Hp})$ in terms of $Time_{Exp_{Re}}(Adv_{Re})$.

[3] The username is included as input to the each random oracle (hash function) call.

To be meaningful, Exp_{Hp} should represent a well studied computational problem, such as factoring, or the computational Diffe-Hellman problem[4].

A *comparative concrete reduction* from hard problem Exp_{Re} to Exp_{Id} and Exp_{Hp}, also includes an algorithm *ReduceId*, defining $Adv_{Id} = ReduceId^{Adv_{Re}}$, and a formula for $Succ_{Exp_{Hp}}(Adv_{Hp})$ in terms of the real-ideal world advantage:

$$Ad = Succ_{Exp_{Re}}(Adv_{Re}) - Succ_{Exp_{Id}}(Adv_{Id}). \tag{1}$$

No Complexity Assumptions: In the concrete framework, no notion of computational indistinguishability is required, and complexity assumptions play a reduced role[5]. Although not required by this framework, a *security parameter* k can be used to calibrate the scheme so that the underlying hard problem is more difficult, and thus the attacker's task more costly.

Random Oracle Disclaimer: Unfortunately, our protocols involve hash functions, yet our security statements do not reduce to the associated hard problems of inverting a hash function on a random input or of finding a collision. Instead, our analysis pretends that the hash functions are replaced with "truly random" functions[1]. As with all random oracle proofs, the security statements we prove, describe most directly properties of a *related* protocol in which all parties must query a distinct cryptographic **trusted third party** to evaluate the hash function. This party, easily implemented with a sorted table, chooses the random function in stages by replying to queries randomly and consistently[6].

4 Two Protocols

Secret Sharing Basis: We now describe a slightly modified version of the protocol[7], and describe our new protocol. Before we begin, we provide the basic intuition and notation common to each. During the registration phase the client splits the password *pass* into shares by choosing a random pad R as the first share, and setting the second share $P = R \oplus pass'$. Later, during the authentication phase, a claimant using password $pass'$, selects a distinct random pad, R', and sets $P' = R' \oplus pass'$. The Blue server computes $\hat{A} = P \oplus P'$, while the Red server obtains $\hat{B} = R \oplus R'$. Clearly $\hat{A} = \hat{B} \iff pass = pass'$.

Relationship with Password Key Exchange: The problem of comparing two values in zero knowledge is known as the *socialist millionaire's problem* [5, 10, 11, 3]. Password based key exchange protocols must solve this problem. The authentication protocols we consider are not key exchange protocols, *per se*, as they already utilitize SSL session keys, but the interaction between the two servers also follows a solution to the socialist millionaire's problem. The reader

[4] Reduction arguments relying on stronger assumptions such as the Decisional Diffe-Hellman assumption, or oracle "gap-assumptions" are somewhat less compelling.

[5] Common complexity-theoretic notions of "negligible function", and "computational indistinguishability" may distract from the focus of a concrete security analysis.

[6] We prefer to explicitly describe the random oracle as a trusted third party so as not to overstate the security implications of a random oracle proof.

is encouraged to compare messages exchanged between Blue and Red (especially in the new scheme) with zero-knowledge key-exchange protocols such as PAK[4].

We also remark that the security of the *three-party protocols* we consider does not follow automatically from a solution to the socialist millionaire's problem. In general, deducing security properties of composed protocol instances is difficult; furthermore, in our particular case, the adversary can influence both $\hat{A}$ and $\hat{B}$.

Notation: The first protocol is from [7] (modifications discussed below), and the new protocol follows the same framework. For easy comparison, we use the same notation and display the two side by side. Let k be an integer, $\mathcal{G}$ be a cyclic group of order q with generator G, and $gp.gen(k)$ an algorithm which generates (a description of) $\mathcal{G}$ [7]. Let $pass$, $pass'$, R, P, R', $P' \in \{0,1\}^k$, e, f be integers in $[1,q]$, $A, B, Y_0, Y_1, Z_0, Z_1 \in \mathcal{G}$; $H_0, H_1 \in \{0,1\}^k$, and $ok_0, ok_1 \in \{0,1\}$. Let w be a function $\{0,1\}^* \to \mathcal{G}$, and let h be a function $\{0,1\}^* \to \{0,1\}^k$, implemented by random oracles W, and H. The symbol $\overset{\$}{\leftarrow}$ denotes a random assignment.

Modified BJKS(k)

```
Parameters
$(\mathcal{G}, q, G) \leftarrow gp.gen(k)$
Registration
$pass \leftarrow passgen()$
$R \overset{\$}{\leftarrow} \{0,1\}^k$
$P \leftarrow R \oplus pass$
Authentication
Client.auth1($pass', U$):
$R' \overset{\$}{\leftarrow} \{0,1\}^k$
$P' \leftarrow R' \oplus pass'$
Blue.auth1($P', U$):
$e \overset{\$}{\leftarrow} [1,q]$
$A \leftarrow w(P \oplus P', U)$
$Y_0 \leftarrow AG^e$
Red.auth1($R', Y_0, U$):
$f \overset{\$}{\leftarrow} [1,q]$
$\star$ $B \leftarrow w(R \oplus R', U)$
$Y_1 \leftarrow BG^f$
$Z_1 \leftarrow (Y_0/B)^f$
$\star$ $H_1 \leftarrow h(Y_0, Y_1, Z_1, R \oplus R', U)$
Blue.auth2($Y_1, H_1$):
$Z_0 \leftarrow (Y_1/A)^e$
$\star$ $H_0 \leftarrow h(H_1, Z_0, P \oplus P', U)$
Red.auth2($H_0$):
$\star$ $ok_1 \leftarrow H_0 \overset{?}{=} h(H_1, Z_1, R \oplus R', U)$
Blue.auth3():
$\star$ $ok_0 \leftarrow H_1 \overset{?}{=} h(Y_0, Y_1, Z_0, P \oplus P', U)$
Client.auth2($ok_0$).
```

New Scheme(k)

```
Parameters
$(\mathcal{G}, q, G) \leftarrow gp.gen(k)$
Registration
$pass \leftarrow passgen()$
$R \overset{\$}{\leftarrow} \{0,1\}^k$
$P \leftarrow R \oplus pass$
Authentication
Client.auth1($pass', U$):
$R' \overset{\$}{\leftarrow} \{0,1\}^k$
$P' \leftarrow R' \oplus pass'$
Blue.auth1($P', U$):
$e \overset{\$}{\leftarrow} [1,q]$
$A \leftarrow w(P \oplus P', U)$
$Y_0 \leftarrow AG^e$
Red.auth1($R', Y_0, U$):
$f \overset{\$}{\leftarrow} [1,q]$
$B \leftarrow w(R \oplus R', U)$
$\star$ $Y_1 \leftarrow G^f$
$Z_1 \leftarrow (Y_0/B)^f$
$H_1 \leftarrow h(Y_0, Y_1, Z_1, R \oplus R', U)$
Blue.auth2($Y_1, H_1$):
$\star$ $Z_0 \leftarrow Y_1^e$
$\star$ $ok_0 \leftarrow H_1 \overset{?}{=} h(Y_0, Y_1, Z_0, P \oplus P', U)$
$\star$ if($ok_0$), $conf \leftarrow P \oplus P'$ else $\emptyset$
$\star$ Red.auth2($ok_0, conf$):
$\star$ $ok_1 \leftarrow R \oplus R' \overset{?}{=} conf$
$\star$ Blue.auth3():
Client.auth2($ok_0$).
```

[7] This description includes an efficient test of membership for $\mathcal{G}$. E.g., $\mathcal{G} = \mathcal{Z}_p^*$.

Modifications: The first scheme described above differs from [7] in computations marked with the symbol $\star$. The main difference is that $P \oplus P'$ and $R \oplus R'$ have been added as hash function inputs. This allows for a less awkward and more efficient security proof. Also, the $\mathcal{G}$ -membership check for Y_0, Y_1 is not depicted, but implicitly assumed to be part of the message parsing.

Although the new scheme resembles the first in form, it is related more naturally to the Diffe-Hellman problem (and to PAK), and thus has a tighter proof. In the new scheme, Y_1 is set to G^f, instead of BG^f, no H_0, is computed, and instead the Blue server sends $R \oplus R'$ back to the Red server as a confirmation message. These further differences are also marked by the symbol $\star$.

5 Adversarial and Computational Problems

Following the framework of Section 3, we now describe the adversarial experiments corresponding to the actual adversary (Exp_{Re}), the ideal-world adversary (Exp_{Id}), and the underlying hard problem (Exp_{Hp}). The concrete security statements tie together the performance of adversaries for three such experiments, quantifying the informal statement "the adversary can't do significantly better than guessing".

ExpReal(k,T)

```
$(\mathcal{G}, q, G) \leftarrow gp.gen(k)$
$pass \leftarrow passgen()$
$R \xleftarrow{\$} \{0,1\}^k$
$P \leftarrow R \oplus pass$
$AdvCorrupt(< sec >)$
for($t = 1$ to $T$)
    $< CompRound() >$
loop
if ($ok_0$=1) return(1) else return(0)
```

ExpIdeal(T)

```
$pass \leftarrow passgen()$
$guess \leftarrow Adv$
$ok \leftarrow guess \overset{?}{=} pass$
return($ok$)
----------------
$Oracle$ Guess(s)
(allowed $T$-1 queries)
$ok \leftarrow s \overset{?}{=} pass$
return($ok$)
```

CDH(k)

```
$(\mathcal{G}, q, G) \leftarrow gp.gen(k)$
$x \xleftarrow{\$} [1,q]$, $X \leftarrow G^x$
$y \xleftarrow{\$} [1,q]$, $Y \leftarrow G^x$
$Z \leftarrow Adv(\mathcal{G}, q, G, X, Y)$
return: $ok \leftarrow Z \overset{?}{=} G^{xy}$
```

CDH-Square(k)

```
$(\mathcal{G}, q, G) \leftarrow gp.gen(k)$
$x \xleftarrow{\$} [1,q]$, $X \leftarrow G^x$
$Z \leftarrow Adv(\mathcal{G}, q, G, X)$
return: $ok \leftarrow Z \overset{?}{=} G^{x^2}$
```

The ideal world adversarial experiment is very simple. It effectively must guess the password in T tries. Clearly, the success probability of any adversary in **ExpIdeal(T)** is less than the sum of the most common T passwords produced by $passgen()$. We denote this probability $GuessProb(T)$.

The hard problem experiment above simply corresponds to the Diffe-Hellman problem for group $\mathcal{G}$. Experiment **CDH-Square(k)** corresponds to the problem of computing $DH(X, X)$ from X, where $DH(G^a, G^b)$ denotes the element G^{ab}.

The real-world experiment is more interesting, and depends on which server is compromised. The secret $< sec >$ revealed during compromise is either P if Blue is compromised, or R if Red is compromised. The per-round interactions ($CompRound()$ above), are described in more detail below with **ExpRealBlue(k,T)** and **ExpRealRed(k,T)**, which reflect the adversary's capabilities listed in Section 2. Separate experiments are given for the new scheme.

5.1 Per Round Interactions

These experiments are designed to follow the framework of in Section 2, except for an additional simplification. Namely, the possibility of the adversary "activating" a user to authenticate with the correct password is treated separately, so that we can assume that the adversary must interact within a session. It is not difficult to see why the adversary will not benefit from this activity. In such a situation $P \oplus P' = R \oplus R'$, and regardless of whether the adversary has compromised Blue or Red, the messages of the other server may be perfectly simulated, so the adversary can learn nothing. Further details are in Appendix D.

CompRedRound

```
Adversary:
$P' \leftarrow AdvClient1()$
Blue.auth1($P'$):
$e \xleftarrow{\$} [1, q]$
$A \leftarrow w(P \oplus P', U)$
$Y_0 \leftarrow AG^e$
Adversary:
$Y_1, H_1 \leftarrow AdvRed.auth1(Y_0)$
Blue.auth2($Y_1, H_1$):
$Z_0 \leftarrow (Y_1/A)^e$
$H_0 \leftarrow h(H_1, Z_0, P \oplus P', U)$
Adversary:
$AdvRed.auth2(H_0)$
Blue.auth3():
$ok_0 \leftarrow H_1 \overset{?}{=} h(Y_0, Y_1, Z_0, P \oplus P', U)$
Adversary:
$AdvClient2(ok_0)$
```

CompRedRoundNew

```
Adversary:
$P' \leftarrow AdvClient1()$
Blue.auth1($P'$):
$e \xleftarrow{\$} [1, q]$
$A \leftarrow w(P \oplus P', U)$
$Y_0 \leftarrow AG^e$
Adversary:
$Y_1, H_1 \leftarrow AdvRed1(Y_0)$
Blue.auth2($Y_1, H_1$):
$Z_0 \leftarrow (Y_1/A)^e$
$ok_0 \leftarrow H_1 \overset{?}{=} h(Y_0, Y_1, Z_0, P \oplus P', U)$
if($ok_0$), $conf \leftarrow P \oplus P'$ else $\emptyset$
Adversary:
$AdvRed2(ok_0, conf)$
Blue.auth3():
Adversary:
$AdvClient2(ok_0)$
```

CompBlueRound

```
Adversary:
$R' \leftarrow AdvClient1()$
$Y_0 \leftarrow AdvBlue1()$
Red.auth1($R', Y_0, U$):
$f \xleftarrow{\$} [1, q]$
$B \leftarrow w(R \oplus R', U)$
$Y_1 \leftarrow BG^f$
$Z_1 \leftarrow (Y_0/B)^f$
$H_1 \leftarrow h(Y_0, Y_1, Z_1, R \oplus R', U)$
Adversary:
$H_0 \leftarrow AdvBlue2(Y_1, H_1)$
Red.auth2($H_0$):
$ok_1 \leftarrow H_0 \overset{?}{=} h(H_1, Z_1, R \oplus R')$
Adversary:
$AdvClient2()$
```

CompBlueRoundNew

```
Adversary:
$R' \leftarrow AdvClient1()$
$Y_0 \leftarrow AdvBlue1()$
Red.auth1($R', Y_0, U$):
$f \xleftarrow{\$} [1, q]$
$B \leftarrow w(R \oplus R', U)$
$Y_1 \leftarrow G^f$
$Z_1 \leftarrow (Y_0/B)^f$
$H_1 \leftarrow h(Y_0, Y_1, Z_1, R \oplus R', U)$
Adversary:
$ok, conf \leftarrow AdvBlue2(Y_1, H_1)$
Red.auth2($ok_0, conf$):
$ok_1 \leftarrow R \oplus R' \overset{?}{=} conf$
Adversary:
$AdvClient2()$
```

These per round interactions have been naturally derived from the real protocols by inserting the adversary's where the compromised server would be active.

6 Concrete Reduction Algorithms

We now present the reduction algorithms, which will ultimately convert the real world adversary into algorithm for solving **CDH(k)** or **CDH-Square(k)**.

6.1 Strategy

We consider adversaries which compromise the Blue and Red servers separately, and we prove each concrete reduction statement in two stages.

In Stage 1, we immediately present the reduction algorithm itself: *Reduce*, which yields a **CDH(k)** adversary $Adv_{Hp} = Reduce^{Adv_{Re}}$ for each real-world adversary Adv_{Re}. Viewing the transcript of a real-world experiment $Exp_{Re}^{Adv_{Re}}$ as a sequence of random variables, we will define (separately for each experiment) an event called an *effective guess* which can be loosely interpreted as a password guess, and we say the event E_{OverT} occurs if there are more than T effective guesses. In this stage we also relate the success $Succ_{Exp_{Hp}}(Adv_{Hp})$ to the probability $Prob[E_{OverT}]$.

The goal of Stage 2, is to compare $Prob[E_{OverT}]$ with the success of the real and ideal-world adversaries. To this end we study the interaction of a real world adversary with a perfect simulator that calls a password guessing oracle. By limiting the number of password guessing oracle calls to T, we obtain an auxiliary reduction algorithm *ReduceId*, which yields an ideal-world Adversary $Adv_{Id} = ReduceId^{Adv_{Re}}$ for each real-world adversary Adv_{Re}. As an ideal world adversary, its success must be bounded by $GuessProb(T)$. The simulation is also constructed such that if all effective guesses are incorrect, the success probability is only $T2^{-k}$. Taken together, this implies that $Succ_{Exp_{Re}}(Adv_{Re}) \leq Prob[E_{OverT}] + GuessProb(T) + T2^{-k}$. Rewriting this relation as

$$Prob[E_{OverT}] \geq Succ_{Exp_{Re}}(Adv_{Re}) - GuessProb(T) - T2^{-k}, \qquad (2)$$

and combining it with the bound of stage 1, we finally relate $Succ_{Exp_{Hp}}(Adv_{Hp})$, to the adversarial advantage (Eq. 1).

6.2 Compromise-Red Reduction (Modified BJKS)

For this experiment, we define an **effective guess** on $\tilde{pass}$ in round t to be the event that for the Y_0, Y_1, H_1, P' sent in round t, oracle H was called on input $(Y_0, Y_1, \tilde{Z}, \tilde{P}, U)$ or on input $(H_1, \tilde{Z}, \tilde{P}, U)$, where $\tilde{P} = P' \oplus R \oplus \tilde{pass}$, $\tilde{A} = W(\tilde{P})$, and $\tilde{Z} = DH(\frac{Y_0}{\tilde{A}}, \frac{Y_1}{\tilde{A}})$. We also let $Q_{\dagger} = Q + 2T$ denote the maximal number of H oracle queries in the entire experiment.

For the BJKS-modified scheme, our aim is reduce to **CDH-Square(k)**. Thus *Reduce* accepts X as input and attempts to compute $DH(X, X)$. *Reduce* calls Adv_{Re} black box, and employs code from the actual protocol, as well as custom versions, *simulations*, of certain functions, generally following the flow of Exp_{Re}. For this reduction, a custom W-oracle embeds X by responding to each new query with X^r for a random $r \in [1, q]$. Next, for one randomly chosen round

$t_0 \in [1, T]$, the usual adversarial round interaction will be replaced with one in which Blue's normal operation is simulated so that (1) $Y_0 = G^a$ has a random known discrete log a, (2) H_0 is set to be equal to a value coinciding with a random H oracle response in the range of indices $[1, Q_\dagger]$, and (3) ok_0 is chosen randomly from $\{0, 1\}$.

Such a simulation is not perfect, but its transcript distribution follows that of an actual real world adversary with probability at least $1/(2Q_\dagger)$. Since t_0 is random, with probability at least $Prob[E_{OverT}]/(2Q_\dagger T)$, two effective guesses will be made on round t_0. In the final stage, two indices in $[1, Q_\dagger]$ are chosen at random. If these indices correspond to H queries of the two effective guesses, these H oracle queries will include the distinct pairs $(\tilde{P}, Z)$ and $(\tilde{P}', Z')$, such that $Z = DH(\frac{Y_0}{\tilde{A}}, \frac{Y_1}{\tilde{A}})$ for $\tilde{A} = W(\tilde{P})$, and $Z' = DH(\frac{Y_0}{\tilde{A}'}, \frac{Y_1}{\tilde{A}'})$ for $\tilde{A}' = W(\tilde{P}')$. Searching through the maximum of $Q+T+2$ calls to the W oracle, we locate the two integers r, and r' such that $\tilde{A} = X^r$ and $\tilde{A}' = X^{r'}$. Provided $r, r', r - r' \neq 0$, the assignment

$$D \leftarrow \{[Z(Y_1/\tilde{A})^{-a}]^{1/r} \ / \ [Z'(Y_1/\tilde{A}')^{-a}]^{1/r'}\}^{1/(r-r')} \tag{3}$$

yields $DH(X, X)$. The final chance of success is $Prob[E_{OverT}]R/(2\{Q_\dagger^3 T)$, where

$$R = 1 - (Q + T + 2 + 2)(Q + T + 2 + 1)/2q \tag{4}$$

lower bounds the chance that all the W-oracle results are distinct and nonzero.

ReduceRed(Adv, T,Q)(X,$\mathcal{G}$,q,G)

```
for $i = 1$ to $Q + T + 2$ :
    $r_i \xleftarrow{\$} [1, q]$, $W_i \leftarrow X^{r_i}$
$t_0 \xleftarrow{\$} [1, T]$, $ind_1 \xleftarrow{\$} [1, Q_\dagger]$
$hrand \xleftarrow{\$} \{0, 1\}^k$
--------------------------------
$pass \leftarrow passgen()$
$R \xleftarrow{\$} \{0, 1\}^k$, $P \leftarrow R \oplus pass$
$AdvCorrupt(R)$
for($t = 1$ to $T$):
    if($t = t_0$) SimulatedRound()
    else CompRedRound()
--------------------------------
$ind_2, ind_3 \xleftarrow{\$} [1, Q_\dagger]$
$(Z_0, \tilde{P}) \leftarrow Hin(ind_2)$
$(Z_0', \tilde{P}') \leftarrow Hin(ind_3)$
$A \leftarrow W(\tilde{P}), r \leftarrow Wseek(A)$
$A' \leftarrow W(\tilde{P}'), r' \leftarrow Wseek(A')$
$D \leftarrow \{\frac{Z(Y_1/A)^{-a}]^{1/r}}{Z'(Y_1/A')^{-a}]^{1/r'}}\}^{1/(r-r')}$
output($D$)
```

W,H Oracle Simulation

```
Oracle W(s) -Adv gets $Q$ queries.
On $i'th$ new query: $w(s) = W_i$.
Oracle H(s) -Adv gets $Q$ queries.
$ind_1$'th new query: $h(s) = hrand$.
Other new queries: $h(s) \xleftarrow{\$} \{0, 1\}^k$
```

Simulated Round

```
Adversary:
$P' \leftarrow AdvClient1()$
Blue.sim1($P'$):
$a \xleftarrow{\$} [1, q]$, $Y_0 \leftarrow G^a$
Adversary:
$Y_1, H_1 \leftarrow AdvRed.auth1(Y_0)$
Blue.sim2($Y_1, H_1$):
$H_0 \leftarrow hrand$
Adversary:
$AdvRed.auth2(H_0)$
Blue.sim3():
$ok_0 \xleftarrow{\$} \{0, 1\}$
Adversary:
$AdvClient2(ok_0)$
```

Passing to stage 2, our goal is to find a bound on $Prob[E_{OverT}]$. This is accomplished with another reduction, $Adv_{Id} = ReduceId^{Adv_{Re}}$ creating an ideal world adversary which makes guesses to a password guessing oracle. Algorithm *ReduceId* is constructed from **ExpRealRed(k,T)**, but replaces the algorithms of Blue, H, and W with simulated algorithms, *which do not directly make use of the password.*

ReduceRedId(Adv, T,Q)

Simulated Round

```
$\mathbf{Blue.sim1}(P')$:
$a \xleftarrow{\$} [1,q]$, $Y_0 \leftarrow G^a$
Adversary:
$Y_1, H_1 \leftarrow AdvRed.auth1(Y_0)$
$\mathbf{Blue.sim2}(Y_1, H_1)$:
for each $h_{in}$
  $(valid, \widetilde{pass}) \leftarrow \mathbf{Valid}(h_{in}, t)$
  if($valid$ AND $\mathbf{IdealGuess}(\widetilde{pass})$)
    $H_0, H_1' \leftarrow \mathbf{Correct}(h_{in})$
    return($H_0$)
$H_0 \leftarrow hrand_t \xleftarrow{\$} \{0,1\}^k$
$H_1' \leftarrow hrand_t' \xleftarrow{\$} \{0,1\}^k$
$\mathbf{Blue.sim3}()$:
$ok_0 \leftarrow H_1' \overset{?}{=} H_1$
```

W,H Oracle Simulation

```
Oracle $\mathbf{W}$(s)
On $i'th$ new query:
$r_i \xleftarrow{\$} [1,q]$ $w(s) = G_i^r$.

Oracle $\mathbf{H}(h_{in})$
for each completed round $t$,
  $(valid, \widetilde{pass}) \leftarrow \mathbf{Valid}(h_{in}, t)$
  if($valid$ AND $\mathbf{IdealGuess}(\widetilde{pass})$)
    $h(h_{in}) \leftarrow hrand_t$
    $h(h_{in}') \leftarrow hrand_t'$
Other new queries:
  $h(h_{in}) \xleftarrow{\$} \{0,1\}^k$

Oracle $\mathbf{IdealGuess}(\widetilde{pass})$)
return ( $\widetilde{pass} \overset{?}{=} pass$)
```

In contrast with Stage 1, Blue's messages will be simulated for every round. The challenge is to simulate Blue's second and third interactions: $\mathbf{Blue.sim2}(Y_1, H_1)$, and $\mathbf{Blue.sim3}()$. Both H_0 and ok_0 must be consistent with the unknown password. This is accomplished by examining all queries made to H so far, to determine if they corresponds to effective guesses. This is done with a test

$$(valid, pass) \leftarrow \mathbf{Valid}(\tilde{Z}, \tilde{P}, t)$$

which returns $valid = 1$, if $\tilde{Z} = DH(\frac{Y_0}{\tilde{A}}, \frac{Y_1}{\tilde{A}})$ for $\tilde{A} = W(\tilde{P})$, and 0 otherwise, and also returns the guessed password $pass = \tilde{P} \oplus P' \oplus R$, when the guess is valid. In fact, this test can be efficiently performed if the discrete log of $\tilde{A}$, and Y_0 are known.[8] The test is used three ways, (1) to ensure each H_0 is consistent with previous H-queries, (2) to ensure each ok_0 is consistent with previous H-queries, (3) to ensure future H-queries are consistent with H_0, ok_0 values of previous rounds.

A slight complication arrises from the fact that an effective guess can correspond to two types of hash calls. To deal with this, the simulator uses the function $\mathbf{Correct}(h_{in})$ which looks up the hash query values $\tilde{Z}$, and $\tilde{P}$, and returns both hash values $H_1' = h(Y_0, Y_1, \tilde{Z}, \tilde{P}, U)$ and $H_0 = h(H_1, \tilde{Z}, \tilde{P}, U)$, once the password is discovered. If the H-oracle has not been called on the inputs corresponding to the correct password, the simulator will choose random responses $hrand_t$ and $hrand_t'$ for the two types of hashes. Later H queries are always

[8] Actually, the efficiency of Adv_{Id} is not important for our argument.

checked, and if one is found to correspond to an effective guesses for a previous round, both types of hash are answered consistently, with $hrand_t$ and $hrand'_t$.

This simulation is perfect if the number of **IdealGuess()** queries is not limited, and the number of queries is distributed exactly as the number of effective guesses made by Adv_{Re} in Exp_{Re}. Thus by the argument above in Section 6.1, we obtain bound (Eq. 2), on $Prob[E_{OverT}]$, so we can can relate $Succ_{Exp_{H_p}}(Reduce^{Adv_{Re}})$ to the advantage Ad defined in (Eq. 1). Letting R be the constant of (Eq. 4), and $Ad' = Ad - T2^{-k}$, we obtain

$$Succ_{Exp_{H_p}}(Reduce^{Adv_{Re}}) \geq \frac{Ad'R}{2TQ_{\dagger}^{3}}. \qquad (5)$$

Furthermore, $Time_{Exp_{H_p}}(Reduce^{Adv_{Re}}) = T_A + T_E + T_R$, where T_E is the time of the experiment Exp_{Re} itself, $T_A = Time_{Exp_{Re}}(Adv_{Re})$, and T_R is the additional time incurred by the reducer itself, which includes a cost proportional to $Qlog(Q)$ to manage the random oracle tables, the $Q + T + 2$ exponentiations for the $W = X^r$ embedding, and the exponentiations required for the final extraction of the candidate $DH(X, X)$ value.

6.3 Compromise-Blue Reduction (BJKS)

This reduction is quite similar to the compromise Red one, except the definition of *effective guess* is somewhat simpler as only one type of H query need be considered. The success of the derived adversary $Reduce^{Adv_{Re}}$ satisfies the same inequality (Eq. 5). Further details on the simulation are given in Appendix A.

6.4 Completing the Reduction to CDH

The reductions, as presented, reduce to the hard problem $\mathbf{CDH-Square(k)}$. In order to further reduce to $\mathbf{CDH(k)}$, we use a well known trick. The equation

$$DH(X,Y)^2 = \sqrt{DH(XY,XY)}/\sqrt{DH(X/Y,X/Y)} \qquad (6)$$

defines a reduction *Reduce*, such that for all $\mathbf{CDH-Square(k)}$ solvers S, with success ϵ completing in time τ, $T = Reduce^S$ is a $\mathbf{CDH(k)}$ solver with success ϵ^2 which completes in time 2τ. Using this approach, a significant loss of success probability results in our reductions. However, when the *decisional* Diffe-Hellman problem is feasible, the situation improves. In this case, we focus on measuring *expected time*, and a $\mathbf{CDH-Square(k)}$ solver S, with expected time τ can be converted to a $\mathbf{CDH(k)}$ solver T with expected time 2τ.

7 Reductions for the New Protocol

The reductions for the new protocol are significantly simpler. First, the apparently small change of setting $Y_1 = G^f$ (instead of $Y_1 = BG^f$), enables a direct

$\mathbf{CDH(k)}$ reduction in the compromise Blue experiment, rather than indirect approach via the $\mathbf{CDH-Square(k)}$ problem. Secondly, using the confirmation message $conf$, allows a direct comparison of the real-world adversary to the password guessing adversary, without even mentioning the **CHD** problem. The difference can easily be seen to be related to the chance of an H-oracle collision.

7.1 Compromise-Red (New Scheme)

For this experiment an **effective guess** on *pass* in round t denotes the event that for the Y_0, Y_1, H_1, P' sent in round t, oracle H was called on input $(Y_0, Y_1, \tilde{Z}, \tilde{P}, U)$ resulting in H_1 where $\tilde{P} = P' \oplus R \oplus pass, \tilde{A} = W(\tilde{P})$, and $\tilde{Z} = DH(\frac{Y_0}{\tilde{A}}, Y_1)$. Note the different requirement for $\tilde{Z}$, and the additional H_1 requirement.

For this experiment we do not need the two stage proof, and instead construct an ideal world adversary directly. The construction of $ReduceID$, follows the same strategy described above in Section 6.2. Specifically, the W oracle is programmed with values of known discrete log, a random $Y_0 = G^a$ is sent with known discrete log a, and consistent values of H_0 are produced by the simulator. All of Blue server messages may be perfectly simulated provided that the oracle **IdealGuess()** is called for each effective guess. As above, this simulation implies Inequality (Eq. 2). However, two effective guesses on a single round would imply

$$h(Y_0, Y_1, \tilde{Z}, \tilde{P}, U) = H_1 = h(Y_0, Y_1, \tilde{Z}, \tilde{P}, U),$$

which is a hash collision. Thus $Prob[E_{OverT}] < S$, where

$$S = 1 - Q(Q-1)/2q \tag{7}$$

is a lower bound on the probability that none of the Q H-oracle results coincide. The resulting bound on the adversarial advantage has nothing to do with $\mathbf{CDH(k)}$.

$$Succ_{Exp_{Re}}(Adv_{Re}) - GuessProb(T) \leq S + T2^{-k}. \tag{8}$$

7.2 Compromise-Blue (New Scheme)

For this experiment an **effective guess** on *pass* round t denotes the event that for the Y_0, Y_1, H_1, P' sent in round t, oracle H was called on input $(Y_0, Y_1, \tilde{Z}, \tilde{P}, U)$, where $\tilde{P} = P' \oplus R \oplus pass$, $\tilde{A} = W(\tilde{P})$, and $\tilde{Z} = DH(\frac{Y_0}{\tilde{A}}, Y_1)$. In this experiment, the maximal number of H-queries is $Q_\dagger = Q + T$.

The reduction $ReduceId$, creating the ideal world adversary, follows the strategy of Section 6.2, yielding the usual Inequality (Eq. 2). However, in contrast with the scheme of [7], the main *Reduce* algorithm, is able to solve $\mathbf{CDH(k)}$ directly. To define *Reduce*, we let X, Y be the $\mathbf{CDH(k)}$ problem instance. Following Section 6.2, the W oracle is programmed with values $B = X_i^r$, and for a randomly chosen round $t_0 \in [1, T]$, Y_1 is set equal to Y, and H_1 is set to be equal to the a'th H-oracle response, for a random index a in the interval $[1, Q_\dagger]$.

In the final stage, two indices in $[1, Q_\dagger]$ are chosen at random. If these indices correspond to the H queries of two effective guesses, these H oracle queries will include two pairs $(\tilde{P}, Z)$ and $(\tilde{P'}, Z')$, such that $Z = DH(\frac{Y_0}{\tilde{A}}, Y_1)$ for $\tilde{A} = W(\tilde{P})$, and $Z' = DH(\frac{Y_0}{\tilde{A'}}, Y_1)$ for $\tilde{A'} = W(\tilde{P'})$. Searching through the (up to $Q+T+2$) W oracle queries, we find the two integers r, and r' such that $A = X^r$ and $A' = X^{r'}$. Provided $r - r' \neq 0$, the formula

$$D \leftarrow [Z/Z']^{r-r'} \tag{9}$$

yields $DH(X,Y)$. The success is at least $Prob[E_{OverT}]R/(Q_\dagger^3 T)$, where R is as in (Eq. 4), (a slightly smaller R suffices), so combining with (Eq. 2), yields

$$Succ_{Exp_{H_P}}(Reduce^{Adv_{Re}}) \geq \frac{Ad'R}{TQ_\dagger^3}. \tag{10}$$

8 Protocol and Proof Variants

Unmodified BJKS: We were still able to find security proofs for the *unmodified* scheme of [7] without the modifications described in Section 4. This analysis is described in Appendix B. The resulting success bound obtained was $\frac{AdR}{TQ_\dagger^5}$.
Using a Decision Oracle: If there is an efficient Diffe-Hellman decision algorithm for $\mathcal{G}$, it makes sense to exploit this in the reduction proofs. For certain elliptic curves, the Weil pairing provides such an efficient procedure. In addition to the technique described in Section 6.4, a decision oracle can be used to improve the simulations in *Reduce*. These techniques are discussed in Appendix C, and we include the expected running time of the CDH adversaries in the summary.
Red Server Notification: In the protocols we have studied, the Red server does not relay the result ok_1 back to the client. The proof for the protocol variant which includes this message requires some small modifications to the simulations.

9 Conclusions

We summarize the success of the derived $\mathbf{CDH(k)}$ Adversaries in this table. This allows a comparison of the reduction efficiency. The final columns show how an adversary can make Q_{DDH} DDH queries to obtain a success rate of $Ad'H/T$, and thus can solve $\mathbf{CDH(k)}$ by with probability near one, by repeating the algorithm *Reps* times. Notice that the difference in reduction "tightness" is significantly more pronounced when a decisional oracle is not available.

Scheme	Corrupted	Time	Success	Q_{DDH}	Reps
BJKS	Blue/Red	$2(T_E + T_A + T_R)$	$(Ad'R/2T)^2/Q_\dagger^{10}$	Q^2	$2T/Ad'R$
BJKS-M	Blue/Red	$2(T_E + T_A + T_R)$	$(Ad'R/2T)^2/Q_\dagger^6$	Q	$2T/Ad'R$
New	Blue	$T_E + T_A + T_R$	$Ad'R/2TQ_\dagger^3$	Q	$T/Ad'R$
	Red	-	-		

The revised scheme presented in this paper is also preferable from several other viewpoints: its security proof is more transparent, the message flows are more directly related to the usual Diffe-Hellman problem, and the server portion of the three party protocol (except the confirmation message) closely relates to the well studied PAK key exchange protocol. The second protocol was also designed to have a security proof, whereas the security proof for scheme [7] was found after its introduction. The preferred approach is to design the protocols concurrently with the proofs.

This work has presented a framework which may be useful for proving *concrete security statements* concerning two-party password-based authentication protocols. More generally, we hope that the approach of presenting explicit *reduction algorithms* instead of reduction proofs will be perceived as adding higher level of transparency to security proofs. Additionally, we hope our approach also illustrates that meaningful concrete security statements can be stated and proved independently of any traditional complexity-theory based foundations[9]. While very useful for feasibility results, complexity theory certainly does not encompass all of cryptography!

10 Acknowledgments

The authors would like to thank Phil MacKenzie for useful discussions, and the anonymous reviewers for comments and corrections.

References

1. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security*, pages 62–73. ACM Press, 1993.
2. S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 72–84. IEEE Press, 1992.
3. F. Boudot, B. Schoenmakers, and J. Traoré. A fair and efficient solution to the socialist millionaires' problem. *Discrete Applied Mathematics*, 111(1-2):23–36, 2001.
4. V. Boyko, P. MacKenzie, and S.Patel. Provably secure password-authenticated key exchange using diffie-hellman. In B. Preneel, editor, *Advances in Cryptology - Eurocrypt '00*, pages 156–, Berlin, 2000. Springer-Verlag. LNCS No. 1807.
5. R. Fagin, M. Naor, and P. Winkler. Comparing information without leaking it. *CACM*, 39(5):77–85, May 1996.
6. W. Ford and B. S. Kaliski Jr. Server-assisted generation of a strong secret from a password. In *Proceedings of the IEEE 9th International Workshop on Enabling Technologies (WETICE)*. IEEE Press, 2000.
7. B. Kaliski and M. Szydlo J. Brainard, A. Juels. Nightingale: A new two-server approach for authentication with short secrets. In *Proceedings of the 12th USENIX Workshop on Security*, pages 1–2. IEEE Computer Society, 2003.
8. D. P. Jablon. Research papers on strong password authentication, 2002. URL: www.integritysciences.com/links.html.

[9] **Corollary**: Assuming computational Diffe-Hellman, we have $Lim_{k \to \infty} Ad(T, k) = 0$.

9. D.P. Jablon. Password authentication using multiple servers. In David Naccache, editor, *Topics in Cryptology - CT-RSA 2001*, pages 344–360. Springer-Verlag, 2001. LNCS no. 2020.
10. M. Jakobsson and A. Juels. Mix and match: Secure function evaluation via ciphertexts. In T. Okamoto, editor, *ASIACRYPT 2000*, pages 162–177. Springer-Verlag, 2000. LNCS no. 1976.
11. M. Jakobsson and M. Yung. Proving without knowing: On oblivious, agnostic, and blindfolded provers. In *CRYPTO '96*, pages 186–200, 1996. LNCS no. 1109.
12. P. MacKenzie, S. Patel, and R. Swaminathan. Password-authenticated key exchange based on rsa. In T. Okamoto, editor, *Advances in Cryptology - Asiacrypt '00*, pages 599–, Berlin, 2000. Springer-Verlag. LNCS No. 1976.
13. P. Mackenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. In M. Yung, editor, *CRYPTO 2002*, pages 385–400. Springer-Verlag, 2002. LNCS no. 2442.
14. M.Bellare, D.Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In B. Preneel, editor, *Advances in Cryptology - Eurocrypt '00*, pages 139–, Berlin, 2000. Springer-Verlag. LNCS No. 1807.

A Compromise Blue Details (BJKS)

This reduction algorithm takes the same form as that of Section 6.2, and we just include it for completeness. For this experiment an **effective guess** on *pass* in round t denotes the event that for the Y_0, Y_1, R' sent in round t, oracle H was called on input $(Y_0, Y_1, \tilde{Z}, \tilde{R}, U)$ where $\tilde{R} = R' \oplus P \oplus pass$, $\tilde{B} = W(\tilde{R})$, and $\tilde{Z} = DH(Y_0/\tilde{B}, Y_1/\tilde{B})$. The problem instance X is embedded via the W oracle, and the Red server messages are simulated for a randomly chosen round t_0. In this one round, the simulation sets $Y_1 = G^a$ to be a random group element with known discrete log a, and imperfectly simulates H_1 setting it to equal one of the H oracle responses. This round's the Red server simulation is shown below. Once all T rounds complete, the reducer chooses two indices at random from $[1, Q_\dagger]$ and obtains the pairs (Z, B) and (Z', B') where $B = X^r$, and $B' = X^{r'}$.

With probability $Prob[E_{OverT}]/Q_\dagger T$, there are two effective guesses in round t_0. If additionally, the two chosen indices correctly correspond to the effective guesses, the desired $DH(X, X)$ may be algebraically derived from Z_0, Z_0', B, B', r, r', a, (see Eq. 3)), provided $r, r', r - r'$ are all non-zero. We conclude that the success is at least $Prob[E_{OverT}]/(\{Q_\dagger^3 TR)$.

Simulated Round - ReduceBlue

```
**Red.sim1**$(R', Y_0)$:
$a \xleftarrow{\$} [1, q]$, $Y_1 \leftarrow G^a$
$H_1 \leftarrow hrand$
**Red.sim2**$(H_0)$:
(nothing)
```

Simulated Round - ReduceBlueID

```
**Red.sim1**$(R', Y_0)$:
$a \xleftarrow{\$} [1, q]$, $Y_1 \leftarrow G^a$
for each $\tilde{H} = H(h_{in})$
    $(valid, \tilde{pass}) \leftarrow$ **Valid**$(h_{in}, t)$
    if($valid$ AND **IdealGuess(**$\tilde{pass}$**)**)
        $H_1 \leftarrow \tilde{H}$, return
$H_1 \leftarrow hrand_t \xleftarrow{\$} \{0, 1\}^k$
**Red.sim2**$(H_0)$:
(nothing)
```

The *ReduceId* algorithm uses the same strategy as Section 6.2, although the simpler definition of effective guess makes the simulation easier (details below). By the same argument above in Section 6.1, we obtain bound (Eq. 2), on $Prob[E_{OverT}]$, so we can can relate $Succ_{Exp_{H_p}}(Reduce^{Adv_{Re}})$ to the advantage Ad (Eq. 1), obtaining for R of (Eq. 4),

$$Succ_{Exp_{H_p}}(Reduce^{Adv_{Re}}) \geq \frac{Ad'R}{TQ_{\dagger}^{3}}. \tag{11}$$

B Unmodified BJKS Scheme

Our modification of the scheme from [7] simply added the values $P \oplus P'$ and $R \oplus R'$ into the hash function inputs. Since the reducer was able to examine the list of H queries, to obtain the $P \oplus P'$ as well as Z_0, the candidate $A = W(P \oplus P')$ values could be computed.

We were still able to find security proofs for the original scheme, without this modification. In this case, the strategy employed by the reducer is to guess the correct W oracle values at random from the entire list of oracle calls, thus obtaining the required $A = G^r$ and $A' = G'^r$, albeit with reduced probability.

The second stage, the comparison with the ideal strategy, was also somewhat more difficult. Following the approach of Section 6.2, the simulator was not able to directly connect the H queries to the password. Instead, the simulation can be made by searching through the list of W oracle queries, and for each image A compute an associated Z. Each Z is compared with the H oracle query list to determine for which passwords "an effective guess" has been made. The resulting simulation is not perfect, since at a later point in the execution, the W oracle may coincidentally produce a random A which corresponds to an effective guess. In such a case, the simulated H_0 or ok_0 may have been inconsistent with the $\{H_0, ok_0\}$ corresponding to the correct password. This imperfect simulation results in a small error term in the inequality comparing the $Prob[E_{OverT}]$, $Succ_{Re}$, and $ExpIdeal(T)$.

An alternate approach to the proof, which enumerates every possible password, may be more efficient for small password dictionaries.

C Using a Decisional Diffe Hellman Oracle

In our proofs, a DDH oracle can be used to check whether a call to the H-oracle is an *effective guess*. This is useful in two ways. First, a *perfect* simulation in *Reduce* can be achieved at a cost of $Q_{\dagger}$ DDH-queries. One query is made for each call to the H-oracle with respect to the chosen round t_0. (Optionally, all T rounds can be used), then the approach of setting H_0 or H_1 randomly is replaced with a strategy which checks all previous H calls for effective guesses.

The second place where the DDH oracle is useful is in the final stage of *Reduce*, where the two Z values are selected. By knowing which H queries are effective guesses, the correct pair can be found when it exists.

Having implemented these changes to *Reduce*, the success can be improved to $Ad'R/T$. Thus, the expected number of repeats of the whole experiment required to solve the hard problem with high probability is approximately $T/Ad'H$ times.

This approaches works for the modified BJKS scheme as well as the new scheme, and for the modified BJKS scheme, passing from $\mathbf{CDH} - \mathbf{Square}(\mathbf{k})$ to $\mathbf{CDH}(\mathbf{k})$ incurs a mere doubling of expected time. However, for the *unmodified* BJKS scheme, it is more difficult to determine an effective guess. By checking each W and each H call, effective guesses may still be determined, but the required number of DDH-queries is now $Q_\dagger^2$. In this case, we also note that the simulation required for *ReduceId* has an extra, small error term.

D Partially Passive Adversaries

To simplify the argument in Section 5.1, we had deferred consideration of adversaries which are active at the time that the valid client authenticates. In some models, the adversary is allowed to "trigger" the client some number of times. We now provide further justification for the claim that this activity will not help the adversary. For convenience, we note the details of the per-round interactions in the cases that Blue or Red is compromised.

CompRedRoundPassive

```
Client.auth1($pass, U$):
$R' \xleftarrow{\$} \{0,1\}^k$
$P' \leftarrow R' \oplus pass'$
Blue.auth1($P'$):
$e \xleftarrow{\$} [1,q]$
$A \leftarrow w(P \oplus P', U)$
$Y_0 \leftarrow AG^e$
Adversary:
$Y_1, H_1 \leftarrow AdvRed.auth1(Y_0, R')$
Blue.auth2($Y_1, H_1$):
..............................
Continues as in Section 5.
```

CompBlueRoundPassive

```
Client.auth1($pass, U$):
$R' \xleftarrow{\$} \{0,1\}^k$
$P' \leftarrow R' \oplus pass'$
Adversary:
$Y_0 \leftarrow AdvBlue1(P')$
Red.auth1($R', Y_0, U$):
$f \xleftarrow{\$} [1,q]$
$B \leftarrow w(R \oplus R', U)$
$Y_1 \leftarrow BG^f$
..............................
Continues as in Section 5.
```

Unlike in Section 5.1, the adversary who corrupts Red is presented with an R' generated by the honest client. As a result, the adversary knows $R \oplus R' = P \oplus P'$, and both the client and Blue server can be perfectly simulated, for example, **Client.sim1:**$R' \xleftarrow{\$} \{0,1\}^k$, **Blue.sim1:** $e \xleftarrow{\$} [1,q]$; $A \leftarrow w(R \oplus R', U)$; $Y_0 \leftarrow AG^e$. This simulation applies to the both the BJKS and new protocols.

Similarly, the adversary who corrupts Blue is presented with an P' generated by the honest client, so this adversary also knows $P \oplus P' = R \oplus R'$, and both the client and Blue server can be perfectly simulated, for example, **Client.sim1:**$P' \xleftarrow{\$} \{0,1\}^k$, **Blue.sim1:** $f \xleftarrow{\$} [1,q]$; $B \leftarrow w(P \oplus P', U)$; $Y_1 \leftarrow BG^e$. Thus, the adversary who prompts the client to enter the correct password, will not learn anything from this round of interaction.