Provably Secure Threshold Password-Authenticated Key Exchange*

Mario Di Raimondo** and Rosario Gennaro***

Abstract. We present two protocols for threshold password authenticated key exchange. In this model, the password is not stored in a single authenticating server but rather shared among a set of n servers so that an adversary can learn the password only by breaking into $t+1$ of them. The protocols require $n > 3t$ servers to work.

The goal is to protect the password against *hackers attacks* that can break into the authenticating server and steal password information. All known centralized password authentication schemes are susceptible to such an attack.

Ours are the first protocols which are provably secure in the standard model (i.e. no random oracles are used for the proof of security). Moreover our protocols are reasonably efficient and implementable in practice. In particular a goal of the design was to avoid costly zero-knowledge proofs to keep interaction to a minimum.

1 Introduction

Password-based authentication is arguably the most deployed mean of authentication in real life applications. The reasons for its wide use are easy to understand: it is mainly its user-friendliness that makes it an attractive choice. Users must remember just a password of their choice and store no other complicated data like long random keys or certificates.

Yet, solutions based on passwords have several security drawbacks. First of all, users tend to choose simple, memorizable passwords. This gives a potential attacker a non-negligible probability of guessing the password and impersonate the user. The most trivial form of this attack (repeatedly try to login until the right password is guessed) can be easily avoided, by careful protocol implementations steps (like disabling an account after a given number of unsuccessful login attempts).

A more dangerous attack is the so-called *off-line dictionary* attack in which the authentication protocol reveals enough information to allow efficient verification of passwords' guesses. In this case the attacker can just perform a search

* Extended Abstract. A full version of this paper is available from `http://www.research.ibm.com/people/r/rosario/tpassword.ps`

** Dipartimento di Matematica e Informatica, Università di Catania, Italy. `diraimondo@dmi.unict.it` – Work done while visiting IBM Research

*** IBM T.J.Watson Research Center. `rosario@watson.ibm.com`

in the password dictionary without ever interacting with the server until he gets the correct password. Thus a major research focus for password-based authentication has been to design protocols that are secure against off-line dictionary attacks. Several such protocols have been presented in a variety of models and with various degrees of security analysis. See below for a detailed list of references.

However, none of the above protocols offers any resistance against *hackers' attacks*, i.e. against attackers who are able to compromise and *break into* the authentication server itself. Indeed in such a case, the attacker will be able to find all password information and mount a dictionary attack against the passwords of *all* users subscribed to that server. Ford and Kaliski [10] identified the problem and its potentially catastrophic consequences and proposed a new model in which the password information is not stored in a single server, but rather in a collection of servers, so that in order to compromise the passwords, the attacker must break into several of them.

In this paper we present two new provably secure protocol in this model of distributed password authentication. Our protocols can be proven secure under the *Decisional Diffie-Hellman Assumption*, in the standard model of computation (i.e. no random oracles are used). The protocols are reasonably efficient and implementable in practice. In particular no costly zero-knowledge proofs are used in order to keep interaction to a minimum.

1.1 Prior Related Work

Research in password-authentication protocols resistant to off-line attacks started with the work of Bellovin and Merritt [3] and continued in [15, 19]. A formal model of security was proposed by Halevi and Krawczyk in [16], where they also presented a provably secure protocol for the case in which the authentication server has a certified public key known to the client. For the case in which the server's key cannot be trusted to be authentic, formal models and provably secure protocols (though in the random oracle model) were presented in [6, 2]. The first (and only currently known) protocol to achieve security without *any* additional setup is that of Goldreich and Lindell [14]. Their protocol is based on general assumptions (i.e., the existence of trapdoor permutations) and should be viewed as a proof of feasibility with respect to obtaining password-based security. Unfortunately, the [14] protocol is very inefficient and thus cannot be used in any practical setting.

We heavily use the efficient protocol by Katz, Ostrovsky and Yung in [23] which is provably secure in the standard model, i.e. without assuming random oracles. It assumes the existence of a publicly known shared random string. We will refer in the following to this protocol as the KOY protocol.

As we mentioned before, Ford and Kaliski [10] put forward the idea of sharing the password information among several servers in order to prevent leaking the passwords to an attacker who is able to break into the authentication server. They present a n-out-of-n solution, i.e., the password information is shared among n servers and they *all* must cooperate to authenticate the user. Although this solution guarantees that the password is secure against an attacker who breaks into $n-1$ servers, it is also less tolerant to random faults. Jablon in [20] improves on the [10], but neither solution comes with a formal proof of security. Independently from our work, Jakobsson, MacKenzie and Shrimpton have presented a t-out-of-n threshold password authentication protocol, which is provably secure in the random oracle model [21].

Thus we can say that our solutions are the first threshold protocols for password authentication which are provably secure in the standard model, i.e. without using random oracles.

This line of research can be thought as applying the tools of *threshold cryptography* to the problem of password authentication. Threshold cryptography aims at the protection of cryptographic secrets, such as keys, by distributing them across several servers in order to tolerate break-ins. See [8, 12] for surveys of this research area. We use and modify several existing threshold cryptography protocols, most importantly Feldman's and Pedersen's VSS protocols [9, 25], their application to discrete-log key generation from [13] and the protocol for multiplication of shared secrets by Abe [1].

1.2 Our Solution in a Nutshell

Our starting point was the KOY protocol from [23]. We present two protocols which are basically t-out-of-n threshold versions of the KOY protocol. As we said before, the schemes are provably secure. The crucial tool in proving security is to show that our protocols are *simulatable*: the adversary's view of the threshold protocol can be recreated by a simulator without having access to the real password.

The basic idea of the protocol is to share the password among n servers using a secret sharing protocol like Shamir's [27] (although we will use a more "redundant" version of Shamir's scheme to achieve a more efficient overall protocol). Then the servers will cooperate to produce the messages that in the KOY protocol a single server was supposed to compute. Notice that this must be done without ever reconstructing the password in a single server, otherwise we expose it to a potential break-in at that server. Some of the tools developed for the full solution are of independent interest and can potentially have more applications. The two protocols differs in the way the client interacts with the collection of the servers.

THE FIRST PROTOCOL. Here we enforce a *transparency* property: to the eyes of the client the protocol should look exactly like a centralized KOY protocol. The client should not be aware that at the server's side the protocol has been implemented in a distributed fashion, nor should he know how many servers are involved. This is achieved by having the client interact with a *gateway* which to the client's eyes will be the authentication server. All the messages exchanged by the client and the gateway will follow the pattern of a regular KOY protocol.

The main advantage of this solution is its efficiency on the client side: the client's work is independent of n and the software installed on the client side does not have to reflect security policies, such as the number n of servers and the threshold t, implemented on the server's side.

Notice however that because of the transparency property, the client will establish a *single* key with the whole collection of servers. This means that this key will be known to the adversary even if he breaks into a single server (for example the gateway). Thus this protocol has the following security properties: (i) if the adversary does not break into any servers, then the protocol is as secure as the original KOY protocol; (ii) if the adversary breaks into less than t servers, then the adversary learns the session key of all the clients who log in during that period, but learns no information about the password of *any* client; (iii) if the adversary breaks into more than t servers then all passwords are compromised.

Notice that we have already made substantial progress with respect to centralized password authentication protocols. Indeed remember that in that case by breaking into a single server the adversary would learn all the passwords stored in that server.

In this case to prove security, it will be enough to show that the view of the adversary when the system is using password pw is identically distributed to the case in which the system is using password $\hat{pw}$ (conditioned to the event that the adversary does not guess the right password, in which case naturally the two views are distinguishable).

THE SECOND PROTOCOL. To strengthen the security properties here we give up on the transparency property. We let the client be aware of the distributed implementation of the servers and in particular of the number n of servers[1]. At the end of this protocol the client will establish n separate keys, one with each server. The adversary will only learn the keys established by the corrupted servers.

Here security is proven by a reduction to the security of the underlying centralized KOY protocol. I.e. we show that if there is an adversary breaking our threshold version, then we can build another adversary which breaks an instance of the centralized original KOY protocol. We are going to use again simulatability to construct this reduction. We have an adversary $\mathcal{A}_{\mathsf{KOY}}$ which is trying to break a centralized version of the KOY protocol. This adversary starts an execution of the threshold adversary $\mathcal{A}_{thresh}$, which we assume is able to break the threshold KOY protocol. This execution is run in a "virtual world" in which all the exchanged messages are simulated. The crucial step in the security proof will be then to "embed" in this virtual world the execution of the centralized KOY protocol which we want to break. Then we can show that a successful attack by $\mathcal{A}_{thresh}$ on this virtual threshold world can be translated in a successful attack in the centralized instance.

[1] We note that this information, namely n, can be transmitted to the client in a handshaking session before starting the actual protocol. Yet the security of our protocol does not rely on the client learning this information in a reliable authenticated matter. I.e. the adversary cannot gain any advantage (besides an obvious denial of service attack) by relying a wrong n to the client.

PROACTIVE SECURITY. In both solutions the password will eventually be exposed if the adversary manages to break into more than t servers. It is possible to apply proactive security solutions [24, 18, 17] to our schemes. In particular we can show that the adversary must break into more than t servers in a single period of time (which is determined according to security policies), in order to gain information about the passwords. The basic idea is to *refresh* the shares of the password after each time period, so that shares from different time periods will be useless in reconstructing the secret. For lack of space this extension will be described in the final version.

AVOIDING ZERO-KNOWLEDGE PROOFS. A clear design goal of our protocols was to avoid the use of costly ZK proofs in order to achieve robustness (i.e. security against a malicious adversary). There are two reasons for this. The first is efficiency: ZK proofs tend to be expensive and increase the amount of interaction required by the protocol. The second is security: password authentication protocols are ran concurrently – the adversary may try to schedule different executions so to try to gain some advantage. Typical ZK proofs are not provably secure when executed concurrently, and modifications to make them concurrently secure tend to make them more complicated and expensive. By avoiding ZK proofs altogether, we simplify the design and facilitate security proofs.

2 Preliminaries

NUMBER THEORY. In the following we denote with p, q two prime numbers such that $q|(p-1)$. We consider the subgroup G_q of Z_p^* of order q and let g be a generator for G_q. All computations are mod$p$ unless otherwise noted.

We are going to assume that the *Decisional Diffie-Hellman Assumption* (DDH) holds in G_q, i.e. no probabilistic polynomial time algorithm given as input three values (g^a, g^b, g^c) can decide if $c = ab \bmod q$ with probability better than 1/2. For a discussion on the DDH see [5].

COMMUNICATION MODEL. As in previous work on password-based authentication, we assume that the communication between the client and the authentication servers, is carried on a basically insecure network. Messages can be tapped and modified by an adversary.

On the other hand we assume that the server's side is implemented via a set of n servers $\mathcal{S}_1, ..., \mathcal{S}_n$. They are connected by a complete network of private (i.e. untappable) point-to-point channels. In addition, the players have access to a dedicated broadcast channel; by dedicated we mean that if server $\mathcal{S}_i$ broadcasts a message, it is received by every other player and recognized as coming from $\mathcal{S}_i$. These assumptions (privacy of the communication channels and dedication of the broadcast channel) allow us to focus on a high-level description of the protocols. However, it is worth noting that these abstractions can be substituted with standard cryptographic techniques for privacy, commitment and authentication.

We assume that the communication channels provide a *partially synchronous* message delivery. That is we assume that the messages sent during the protocol, are received by their recipients within some fixed time bound. Notice that this will allow a malicious adversary to wait for the messages of the honest servers in a given round before sending her own. In the cryptographic protocols literature this is also known as a *rushing* adversary.

2.1 Threshold Password Authenticated Key Exchange

For the centralized case we use the formal security model for password key exchange protocols from [6, 2] (or see full version). Here we explain how to extend it for the case of a distributed server.

SECURITY FOR THE DISTRIBUTED CASE. In the distributed case the goal of the adversary will remain the same (i.e. try to learn some information about a session key). But we will modify somewhat the adversary powers. First of all the adversary will not have total control of the internal network of the servers. We assume that the servers have some authentication mechanism already in place in order to create secure channels between them. Moreover we give the adversary the power to break into servers. This will allow her to see their internal state and totally gain control of that server. That means that the adversary will receive all the messages sent to that server and will reply for him. Notice that we are allowing a *malicious* adversary which can modify the behavior of the server in any way. We bound the number of servers that the adversary can corrupt with t. We assume the adversary to be *static*, i.e. the set of corrupted players is decided in advance (known techniques can be used to make this protocol secure against adaptive adversaries [7, 11, 22]).

TRANSPARENT PROTOCOLS. We say that a distributed protocol is *transparent*, if the client's work is independent of n. Thus to the client the distributed protocol looks like a centralized one. At the end of a transparent protocol the client shares a key sk with the n servers. In this case our definition of security imposes two conditions:

1. If the adversary does not corrupt any servers, then the protocol is secure according to the centralized definition;
2. If the adversary corrupts from 1 to t servers, then all we ask is that the adversary gains no information about the clients' passwords. I.e. the probability of guessing the right password remains $1/|\mathsf{Dict}|$. Formally we define as $view(pw)$ the view of the adversary during a protocol using password pw. The requirement is that for any two passwords $pw, \hat{pw}$ the two views $view(pw)$ and $view(\hat{pw})$ can be distinguished with at most probability $Q/|\mathsf{Dict}|$.

Notice that condition (2) is much weaker than full security, but it is the best we can hope for a transparent protocol. A protocol satisfying the above conditions is called *transparently secure*. Such protocols guarantee the secrecy of the passwords even against compromises of the servers. However they do not guarantee that the session key will be secret from an adversary who breaks into a server.

In practice this is not a major drawback. Usually servers' compromises are brief and quickly discovered: the adversary can easily be "kicked out" of the compromised server by operations such as rebooting and reloading of key software components from a trusted basis. A transparently secure protocol guarantees that during these briefs breaches only the session keys of the clients who log in are compromised. No long-term information, in particular the password, is leaked.

Since transparent protocols are usually much more efficient for the clients, we think that even with this relaxed security notion, this is a useful model.

General Protocols. In this case we allow the parameters n, t to be known to the client who will effectively interact with all n servers. Here the output of the protocol will be n independent session keys: the client will share one key with each server.

The definition of security is obtained by just extending the oracle calls Reveal, Test to specify the index of the server for which the adversary wants to see the relevant session key. The security goal (i.e. the advantage of the adversary remains roughly the same as in the trivial attack of guessing the password) remains unchanged.

2.2 The KOY Protocol

In this section we briefly recall the KOY protocol from [23]. The protocol can be proven secure (according to the above definition) under the DDH Assumption. We will not need their proof of security for our case since we will reduce the security of our distributed version to the security of the original KOY protocol. For details the reader is referred to [23].

The protocol uses some public information which is composed by: two primes p, q, such $q|(p-1)$; five elements of order q in Z_p^*, f, g, h, c, d, and a universal one-way hash function $\mathcal{H}$. The protocol appears in Figure 1.

2.3 VSS Protocols

We assume the reader is familiar with Shamir's Secret Sharing protocol [27].

Feldman's VSS. Feldman's Verifiable Secret Sharing (VSS) protocol [9], extends Shamir's secret sharing method in a way to tolerate a malicious adversary which corrupts up to $\frac{n-1}{2}$ servers *including the dealer*.

Feldman's VSS uses the parameters p, q, g as defined earlier. Like in Shamir's scheme, the dealer generates a random t-degree polynomial $S(\cdot)$ over Z_q, s.t. $S(0) = s$, and transmits to each server $\mathcal{S}_i$ a share $s_i = S(i) \bmod q$. The dealer also broadcasts values $V\!S_k = g^{a_k}$ where a_k is the k^{th} coefficient of $S(\cdot)$. This will allow the players to check that the values s_i really define a secret by checking that

$$g^{s_i} = \prod_k (V\!S_k)^{i^k} \bmod p \tag{1}$$

KOY

Public Information: $p, q, f, g, h, c, d, \mathcal{H}$

1. The client $\mathcal{C}$ generates a key pair $(\mathsf{VK}, \mathsf{SK})$ for a one-time signature scheme. Then he chooses $r' \in_R Z_q$ and computes $A = f^{r'}$, $B = g^{r'}$, $C = h^{r'} f^{pw}$, $\alpha = \mathcal{H}(\mathcal{C}|\mathsf{VK}|A|B|C)$ and $D = (cd^{\alpha})^{r'}$.
   The client $\mathcal{C}$ then sends to the server $\mathcal{S}$ the following message: $\mathcal{C}|\mathsf{VK}|A|B|C|D$
2. The server $\mathcal{S}$ chooses $x, y, z, w, r \in_R Z_q$, recomputes the hash $\alpha' = \mathcal{H}(\mathcal{C}|\mathsf{VK}|A|B|C)$ and then computes: $E = f^x g^y h^z (cd^{\alpha})^w$, $F = f^r$, $G = g^r$, $I = h^r f^{pw}$, $\beta = \mathcal{H}(\mathcal{S}|E|F|G|I)$ and $J = (cd^{\beta})^r$.
   The server $\mathcal{S}$ then sends to the client $\mathcal{C}$ the following message: $\mathcal{S}|E|F|G|I|J$
3. The client $\mathcal{C}$ chooses $x', y', z', w' \in_R Z_q$ and computes: $\beta' = \mathcal{H}(\mathcal{S}|E|F|G|I)$ and $K = f^{x'} g^{y'} h^{z'} (cd^{\beta})^{w'}$.
   He then signes the message $(\beta|K)$ with the key SK, thus obtaining the signature $\mathsf{Sig} = \mathsf{Sign}_{\mathsf{SK}}(\beta|K)$.
   He finally sends to $\mathcal{S}$ the message: $K|\mathsf{Sig}$.
   The client computes the session key as follows: $sk_{\mathcal{C}} = E^{r'} F^{x'} G^{y'} (If^{-pw})^{z'} J^{w'}$
4. The server $\mathcal{S}$ checks the signature using the key VK and if it is correct he then computes the session key as follows: $sk_{\mathcal{S}} = K^r A^x B^y (Cf^{-pw})^z D^w$.

Fig. 1. The KOY Protocol

If the above equation is not satisfied, server $\mathcal{S}_i$ asks the dealer to reveal his share (we call this a *complaint*.) If more than t players complain then the dealer is clearly bad and he is disqualified. Otherwise he reveals the share s_i matching Equation (1) for each complaining $\mathcal{S}_i$.

Equation (1) also allows detection of incorrect shares s_i' at reconstruction time. Notice that the value of the secret is only computationally secure, e.g., the value $g^{a_0} = g^s$ is leaked. However, it can be shown that an adversary that learns t or less shares cannot obtain any information on the secret s beyond what can be derived from g^s. This is good enough for some applications, but it is important to notice that it does not offer "semantic security" for the value s. In particular if s is a password, then revealing g^s opens the door to an off-line dictionary search.

We will use the following notation to denote the execution of a Feldman's VSS protocol.

$$\mathsf{Feldman\text{-}VSS}[s](g) \xrightarrow[t,n]{S} [s_i](\mathcal{V}\!S_k)$$

Pedersen's VSS. We now recall a VSS protocol, due to Pedersen [25], that provides information theoretic secrecy for the shared secret. In addition to the parameters p, q, g, it uses an element $h \in G_q$, and the discrete log of h in base g is unknown (and assumed hard to compute).

The dealer first chooses two t-degree polynomials $S(\cdot), \tilde{S}(\cdot)$, with random coefficients over Z_q, subject to $S(0) = s$, the secret. The dealers sends to each server $\mathcal{S}_i$ the values $s_i = S(i) \bmod q$ and $\tilde{s}_i = \tilde{S}(i) \bmod q$. The dealer then commits to each coefficient of the polynomials S and $\tilde{S}$ by publishing the values $\mathcal{VS}_k = g^{a_k} h^{b_k}$, where a_k (resp. b_k) is the k^{th} coefficient of S (resp. $\tilde{S}$).

This allows the players to verify the received shares by checking that

$$g^{s_i} h^{\tilde{s}_i} = \prod_k (\mathcal{VS}_k)^{i^k} \bmod p \tag{2}$$

The players who hold shares that do not satisfy the above equation broadcast a complaint. If more than t players complain the dealer is disqualified. Otherwise the dealer broadcasts the values s_i and $\tilde{s}_i$ matching the above equation for each complaining player $\mathcal{S}_i$.

At reconstruction time the players are required to reveal both s_i and $\tilde{s}_i$ and Equation (2) is used to validate the shares. Indeed in order to have an incorrect share σ_i' accepted at reconstruction time, it can be shown that player $\mathcal{S}_i$ has to compute the discrete log of h in base g.

Notice that the value of the secret is unconditionally protected since the only value revealed is $\mathcal{VS}_0 = g^s h^{b_0}$ (it can be seen that for any value s' there is exactly one value b_0' such that $\mathcal{VS}_0 = g^{\sigma'} h^{b_0'}$ and thus $\mathcal{VS}_0$ gives no information on s).

We will use the following notation to denote an execution of Pedersen's VSS:

$$\textsf{Pedersen-VSS}[s, \tilde{s}](g, h) \xrightarrow[t,n]{S,\tilde{S}} [s_i, \tilde{s}_i](\mathcal{VS}_k)$$

Joint Sharings. To avoid the use of a trusted dealer, it is possible to generate jointly a random shared secret. The idea is that each player runs a copy of Shamir's or Pedersen's protocols with a random secret. The resulting shared secret is the sum of the secrets shared by each player (in the case of Pedersen's VSS the sum is taken only over the players that were not disqualified as dealers). Notice that by adding up the shares that he received, each player will retain a share of the final secret. Similarly in the case of Pedersen's VSS the verification information $\mathcal{VS}_k$ for the final secret can be obtained by multiplying the commitments published by each player.

We will later require the following notation:

$$\textsf{Joint-Pedersen-RVSS}(g, h) \xrightarrow[t,n]{S,\tilde{S}} [s_i, \tilde{s}_i, \mathcal{TS}](\mathcal{VS}_k, \mathcal{QUAL})\{s\}$$

which follows the same syntax of the basic Pedersen's protocol (with all the parameters referred to the final secret), except that we are adding the values $\mathcal{TS}$ and $\mathcal{QUAL}$.

$\mathcal{TS}$ is the *transcript* of the n VSS's executed by each player. We need to save this information, because it will be useful later on (i.e. our protocol does not just use the final secret and "forgets" how it was created). In particular $\mathcal{T}_S$ includes a private part for each player $\mathcal{S}_i$ which consists of all the shares he received, plus a public part which consists of all the verification informations for the n VSS's. $QUAL$ is the set of players which were not disqualified during the n VSS's.

2.4 Abe's Multiplication Protocol

The last tool we use is Abe's multiplication protocol [1]. This protocol takes as input the transcripts of two Pedersen-VSS protocols for secrets r, s and outputs a new sharing of the product rs. This new sharing can be seen in the form of a Joint-Pedersen of the value rs. See [1] for details.

We are going to use the following notation for this protocol:

$$\mathsf{Abe\text{-}Mult}[\mathcal{T}_S, \mathcal{T}_R](g,h) \xrightarrow[t,n]{} [\chi_i, \tilde{\chi}_i, \mathcal{T}_X](V_{\chi k}, QUAL)\{\chi = rs\}$$

which with regards to the output part follows the same syntax as Joint-Pedersen-RVSS (i.e. the various terms have the same meaning).

This is the only subprotocol that requires $n > 3t$. We notice that standard multiparty computation techniques could be used to implement multiplication with $n > 2t$. But the resulting protocols would be much more complicated. We decided to focus on simplicity rather than optimal threshold.

3 Shared Exponentiation of Secrets

This section describes a new tool which we use as part of our main solution. This protocol is of independent interest and may find applications in other cases.

If we look at the first message sent by the centralized server in the KOY protocol, we see that it includes the value $E = f^x g^y h^z (cd^\alpha)^w$, where x, y, z, w are random secrets. Thus, if we want to distribute this functionality we need a protocol that can output the value $g_1^{s_1} \cdots g_m^{s_m} \bmod p$, where the s_i's are random secrets which have been generated jointly by the servers.

Let's start for simplicity for the case $m = 1$, i.e. the players need to generate a random secret s shared among them and publicize the value g^s. A solution is presented in [13]: they called their protocol DKG (for Distributed Key Generation) since its main application is the distributed generation of keys for discrete log based cryptosystems. The DKG protocol is basically a Joint-Pedersen-RVSS with basis g, h, which generates the shared secret s. It is then followed by a second phase in which each player performs a Feldman's VSS with basis g with the same polynomial used for the Joint-Pedersen-RVSS. This second phase will allow the players to calculate g^s securely[2].

We modify the DKG protocol so that it works with any combination of basis (i.e. the basis used in the Joint-Pedersen-RVSS need not be equal to the ones

[2] In [13] it is also explained why a joint version of Feldman's VSS is not sufficient, since it allows the adversary to bias the distribution of the secret s.

Shared-Exp

Input : the transcripts $\mathcal{T}_R, \mathcal{T}_S$

1. Every server $\mathcal{S}_i$ $(i \in QUAL)$ computes the values:

$$E_{ik} = g^{\alpha_{ik}} \cdot h^{a_{ik}} \bmod p \text{ for } k \in [0..t]$$

and broadcasts them.

2. Every server $\mathcal{S}_j$ $(j \in QUAL)$ checks that: for every $i \in QUAL$ it holds that

$$g^{r_{ij}} h^{s_{ij}} \stackrel{?}{=} \prod_{k=0}^{t} (E_{ik})^{j^k} \bmod p \tag{3}$$

If the checks fail for some i, then $\mathcal{S}_j$ complains against $\mathcal{S}_i$, by revealing in broadcast the values $r_{ij}, \tilde{r}_{ij}, s_{ij}, \tilde{s}_{ij}$. These values do not satisfy the above check but satisfy the Pedersen's VSS checks (since $\mathcal{S}_i \in QUAL$).

3. If $\mathcal{S}_i$ receives at least one valid complaint, then the servers run the reconstruction phase of $\mathcal{S}_i$'s VSS and obtain the values α_{ik}, a_{ik} and E_{ik} for $k \in [0..t]$.
4. Now each player can compute

$$g^r h^s = \prod_{i \in QUAL} E_{i0} \bmod p$$

Fig. 2. Two-secret version of the protocol for exponentiation of shared secrets

used to exponentiate the secrets). Moreover we extend DKG to work with any number of secrets. In the future we will refer to an execution of this protocol with the following notation:

$$\text{Shared-Exp}[\mathcal{T}_{S_1}, \ldots, \mathcal{T}_{S_m}](g_1, \ldots, g_m) \rightarrow (g_1^{s_1} \cdots g_m^{s_m})$$

The protocol appears in Figure 2. For simplicity we limit ourselves to two secrets r, s, with basis g, h (i.e. we compute $g^r h^s$). It should be clear how to extend it to any number of secrets. The description of the protocol assumes that two Joint-Pedersens for the secrets r, s have already been performed. Thus the input of the players are the transcripts $\mathcal{T}_R, \mathcal{T}_S$. We recall exactly what these transcripts include:

$$\mathcal{T}_R = \{\text{private to } \mathcal{S}_i \ (i \in QUAL) : \{\alpha_{ik}\}, \{r_{ji}, \tilde{r}_{ji}\}; \text{public} : \{\mathcal{V}_{R_{ik}}\}\}$$

$$\mathcal{T}_S = \{\text{private to } \mathcal{S}_i \ (i \in QUAL) : \{a_{ik}\}, \{s_{ji}, \tilde{s}_{ji}\}; \text{public} : \{\mathcal{V}_{S_{ik}}\}\}$$

where a_{ik} (resp. α_{ik}) are the coefficients of the sharing polynomial used by $\mathcal{S}_i$ during the joint generation of s (resp. r). The set $QUAL$ can be taken to be the intersection of the qualified set from each previous Joint-Pedersen protocol. In the first step of the protocol each server $\mathcal{S}_i$ reveals the values E_{ik} which are of the form $g^{\alpha_{ik}} h^{a_{ik}}$. This will allow the players to check that they are correct by matching them with the shares they hold from the previous Pedersen-VSS. If $\mathcal{S}_i$ gives out incorrect E_{ik}'s then we expose his component in the Joint-VSS by reconstruction. The desired value is then the product of the E_{i0}'s.

The protocol Shared-Exp is secure against an adversary which corrupts up to $\frac{n-1}{2}$ of the servers (in other words we require $n > 2t$). Security is defined in the usual terms: the execution of the protocol should not reveal any information about r, s beyond the output value $g^r h^s$. This is formally proven by a simulation argument.

Lemma 1. *If $n > 2t$, then the protocol* Shared-Exp*, is a secure protocol which on input the transcripts of m* Joint-Pedersen-RVSS *for secrets $s_1, \ldots, s_m$, and m elements of G_q $g_1, \ldots, g_m$, computes $\prod_{i=1}^{m} g_i^{s_i}$.*

4 The first protocol

We now have all the tools to present our first protocol. As we said in the Introduction, this protocol will be transparent to the client. $\mathcal{C}$ will interact with just one server, the *gateway*, while in the background the servers will cooperate to produce KOY-like messages for the client. The gateway can be any of the servers, thus we are not creating a single point of failure in the network (since if one gateway fails, another one can always take its place). We now give an informal description of the protocol.

How to share the password. We assume that the password has already been "installed" in a shared form across the servers. This could be done by having a trusted process (for example the client) share the password via Pedersen-VSS. However for efficiency reason we will use a more redundant sharing. We simulate a Joint-Pedersen-RVSS which has as result the password itself (this technique is somewhat reminiscent of the share-backup technique of [26], though we use it for different purposes).

That is, the password pw is first split in n random values $pw_1 \ldots pw_n$ such that $pw = pw_1 + \ldots + pw_n \bmod q$ (we assume that we can always encode pw as a member of Z_q). The value pw_i is given to server $\mathcal{S}_i$. Then each pw_i is shared via a (t, n) Pedersen-VSS. The sharing polynomials are also given to server $\mathcal{S}_i$. Each server $\mathcal{S}_j$ will save the whole transcript of the execution. By summing up all the shares received in the n previous VSS's, server $\mathcal{S}_j$ will obtain values $p_j, \tilde{p}_j$ which are shares on a (t, n) Pedersen-VSS of pw. All this work can be performed locally by the trusted process, and then the sharing information can be forwarded to the servers.

Notice that in this Joint-VSS the qualified set of players *QUAL* is the whole set, since this Joint-VSS is simulated by a trusted process.

The authentication protocol. Once the password is shared, the actual protocol goes as follows. We follow the pattern of a KOY protocol closely (so the reader may want to keep Figure 1 handy while reading this description). First the client sends to the gateway the message $\mathcal{C}|\mathsf{VK}|A|B|C|D$.

On receipt of this message the servers perform five Joint-Pedersen-RVSS to select random values r, x, y, z, w and keep shares of them. Then using the Shared-Exp protocol (four times) they can compute $E = f^x g^y h^z (cd^\alpha)^w$, $F = f^r$, $G = g^r$, $I = h^r f^{pw}$. This is where the redundant sharing of the password helps, since it allows us to compute I via a simple invocation of the Shared-Exp protocol[3]. Once E, F, G, I are public the servers can compute β and then, via another invocation of Shared-Exp, the value J.

At the same time the servers also perform an invocation of Abe-Mult to compute a sharing of the value $z \cdot pw$.

The gateway forwards the message $\mathcal{S}|E|F|G|I|J$ to the client (we assume that $\mathcal{S}$ is the "name" of the collective authentication server for the client). Following the KOY protocol the client will answer with $K|\mathsf{Sig}$.

The servers will first authenticate the signature and, if valid, they need to compute the session key $sk_{\mathcal{S}} = K^r A^x B^y C^z D^w f^{-zpw}$. This can be computed via another Shared-Exp invocation, since the servers have all the relevant sharings.

A detailed exposition of the protocol appears in Figure 3.

About Efficiency. It should be clear that the protocols in steps (3a, 3b, 7b) can be performed in parallel, thus reducing the number of rounds.

Moreover we observe that the whole of step 3a, plus the computations of F, G, I can be done *offline* by the servers, before starting an actual interaction with the client. By saving the intermediate results and carefully keeping a synchronized state, the servers can then use these values already prepared when the client initiates a protocol. This reduces to a minimum the amount of computation and communication that the servers need to perform during the actual authentication protocol. Notice that no ZK proofs are used anywhere in the protocol.

Theorem 1. *If $n > 3t$ and the DDH Assumption holds, then the protocol* Dist-KOY1 *is a transparently secure distributed password authentication protocol.*

5 The second protocol

In this section we show a protocol which is not transparent, so is less efficient than the previous one, but achieves a higher level of security. The client will establish n separate session keys, one with each server $\mathcal{S}_i$. We will prove that the adversary can only learn the keys established with the corrupted servers, and has no information about the keys held by the good servers.

The basic idea is to run n separate KOY protocols, one between the client and each server. The client will use the same password for each execution. Since the password is shared at the servers' side, they will cooperate in the background to produce the messages in each execution. Here is an informal description of this protocol.

[3] See the full version for a non-essential technical remark about the computation of I

Dist-KOY1

Public Information: $p, q, f, g, h, c, d, l, \mathcal{H}$
Input for Client $\mathcal{C}$: the password $pw \in Z_q$
Input for servers: T_P: the output of the simulated Joint-Pedersen-RVSS for the password pw.

1. The client runs the first step of the KOY protocol. It results in the message: $\mathcal{C}|\mathsf{VK}|A|B|C|D$ sent to the gateway.
2. The gateway broadcasts to all the servers the previous message. The servers compute
$$\alpha = \mathcal{H}(\mathcal{C}|\mathsf{VK}|A|B|C) \text{ and } M = cd^{\alpha} \bmod p$$
3. (a) The servers $\mathcal{S}_i$ perform the following:
$$\text{Joint-Pedersen-RVSS}(f, l) \xrightarrow[t,n]{R,\tilde{R}} [r_i, \tilde{r}_i, T_R](V_{R_k})\{r\}$$
$$\text{Joint-Pedersen-RVSS}(f, l) \xrightarrow[t,n]{X,\tilde{X}} [x_i, \tilde{x}_i, T_X](V_{X_k})\{x\}$$
$$\text{Joint-Pedersen-RVSS}(f, l) \xrightarrow[t,n]{Y,\tilde{Y}} [y_i, \tilde{y}_i, T_Y](V_{Y_k})\{y\}$$
$$\text{Joint-Pedersen-RVSS}(f, l) \xrightarrow[t,n]{Z,\tilde{Z}} [z_i, \tilde{z}_i, T_Z](V_{Z_k})\{z\}$$
$$\text{Joint-Pedersen-RVSS}(f, l) \xrightarrow[t,n]{W,\tilde{W}} [w_i, \tilde{w}_i, T_W](V_{W_k})\{w\}$$
We denote by $QUAL$ the intersection of all the qualified sets in the above five protocols.
(b) The servers perform the following:
$$\text{Shared-Exp}[T_X, T_Y, T_Z, T_W](f, g, h, M) \rightarrow (E = f^x g^y h^z M^w)$$
$$\text{Shared-Exp}[T_R](f) \rightarrow (F = f^r)$$
$$\text{Shared-Exp}[T_R](g) \rightarrow (G = g^r)$$
$$\text{Shared-Exp}[T_R, T_P](h, f) \rightarrow (I = h^r f^{pw})$$
$$\text{Abe-Mult}[T_Z, T_P](f, l) \xrightarrow[t,n]{} [\chi_i, \tilde{\chi}_i, T_\chi](V_{\chi_k}, QUAL)\{\chi = z \cdot pw\}$$
The servers can now compute
$$\beta = \mathcal{H}(\mathcal{S}|E|F|G|I) \text{ and } N = cd^{\beta} \bmod p$$
where $\mathcal{S}$ is the "name" of the collective authentication server.
(c) The servers can now compute:
$$\text{Shared-Exp}[T_R](N) \rightarrow (J = N^r)$$
4. The gateway sends to the client the message: $\mathcal{S}|E|F|G|I|J$ as in a regular KOY protocol.
5. The client $\mathcal{C}$ follows the KOY protocol and answers with $K|\mathsf{Sig}$.
6. The gateway sends to all the servers the previous message $K|\mathsf{Sig}$
7. (a) The servers can verify the signature and if it is not correct, it stops.
(b) The servers perform the following:
$$\text{Shared-Exp}[T_R, T_X, T_Y, T_Z, T_W, T_\chi](K, A, B, C, D, f^{-1}) \rightarrow (sk_{\mathcal{S}})$$
to compute the session key $sk_{\mathcal{S}} = K^r A^x B^y C^z D^w f^{-z \cdot pw}$.

Fig. 3. Transparent Distributed Version of the KOY protocol

The client initiates n instances of the KOY protocol, sending to the servers the message $\mathcal{C}|\mathsf{VK}_j|A_j|B_j|C_j|D_j$ for $j = 1 \dots n$.

The servers will send back n KOY messages. That is for each server $\mathcal{S}_j$ the servers will jointly select random values r_j, x_j, y_j, z_j, w_j and send back to the client $E_j = f^{x_j} g^{y_j} h^{z_j} (cd^{\alpha})^{w_j}$, $F_j = f^{r_j}$, $G_j = g^{r_j}$, $I_j = h^{r_j} f^{pw}$ and $J_j = (cd^{\beta_j})^{r_j}$. As in the first protocol the servers will also perform Abe's multiplication protocol to get a sharing of $z_j \cdot pw$.

The servers also perform an additional joint sharing for a random value s_j. This value will be sent privately to server $\mathcal{S}_j$ (i.e. the servers will send him the shares and he will reconstruct it).

The n messages $\mathcal{S}_j|E_j|F_j|G_j|I_j|J_j$ are sent to the client. He will answer with n messages $K_j|\mathsf{Sig}_j$, which follow the KOY protocol.

The servers will first authenticate the signatures and, if valid, they will cooperate to compute "masked" session keys $mask_j = l^{s_j} K_j^{r_j} C_j^{z_j} A_j^{x_j} B_j^{y_j} D_j^{w_j} f^{-z_j pw}$. As before this is another invocation of the Shared-Exp protocol.

Server $\mathcal{S}_j$ will then set the secret key as $sk_j = mask_j \cdot l^{-s_j}$.

A detailed exposition of the protocol appears in Figure 4. We omit the steps regarding the gateway. As in the previous protocol the password pw is installed in a shared form via a Joint-Pedersen-RVSS.

Theorem 2. *If $n > 3t$ and the DDH Assumption holds, then the protocol* Dist-KOY2 *is a secure distributed password authentication protocol.*

Acknowledgments: Thanks to Phil MacKenzie for invaluable comments at various stages of this project, Yehuda Lindell for enlightening conversations about password protocols and Stas Jarecki for very helpful suggestions.

References

1. M. Abe, Robust distributed multiplication without interaction. In *CRYPTO '99*, Springer LNCS 1666, pp. 130–147, 1999.
2. M. Bellare, D. Pointcheval and P. Rogaway, Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT 2000*, Springer LNCS 1807, pp. 139–155, 2000.
3. S. M. Bellovin and M. Merritt, Encrypted Key Exchange: Password based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Research in Security and Privacy*, IEEE Press, pp. 72–84, 1992.
4. M. Ben-Or, S. Goldwasser, and A. Wigderson, Completeness Theorems for Non-cryptographic Fault-Tolerant Distributed Computations. In 20^{th} *Annual Symposium on the Theory of Computing*, ACM Press, pp. 1–10, 1988.
5. D. Boneh, The Decision Diffie-Hellman Problem. In *Third Algorithmic Number Theory Symposium*, Springer LNCS 1423, pp. 48–63, 1998.
6. V. Boyko, P. D. MacKenzie, and S. Patel, Provably secure password-authenticated key exchange using Diffie-Hellman. In *EUROCRYPT 2000*, Springer LNCS 1807, pp. 156–171, 2000.
7. R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin, Adaptive Security for Threshold Cryptosystems. In *CRYPTO '99*, Springer LNCS 1666, pp. 98–115, 1999.

Dist-KOY2

Input: as in Dist-KOY1.

1. The client runs n times the first step of the KOY protocol. It results in the messages: $\mathcal{C}|\mathsf{VK}_j|A_j|B_j|C_j|D_j$ for $j = 1 \dots n$ sent to the gateway.
2. The servers compute
$$\alpha_j = \mathcal{H}(\mathcal{C}|\mathsf{VK}_j|A_j|B_j|C_j) \text{ and } M_j = cd^{\alpha_j} \bmod p$$
Then for each $j = 1 \dots n$ the following steps are performed:
   (a) The servers perform
$$\mathsf{Joint\text{-}Pedersen\text{-}RVSS}(f,l) \xrightarrow[t,n]{R_j,\tilde{R}_j} [r_{ji}, \tilde{r}_{ji}, \mathcal{T}_{R_j}](\mathcal{V}_{R_{jk}})\{r_j\}$$
$$\mathsf{Joint\text{-}Pedersen\text{-}RVSS}(f,l) \xrightarrow[t,n]{X_j,\tilde{X}_j} [x_{ji}, \tilde{x}_{ji}, \mathcal{T}_{X_j}](\mathcal{V}_{X_{jk}})\{x_j\}$$
$$\mathsf{Joint\text{-}Pedersen\text{-}RVSS}(f,l) \xrightarrow[t,n]{Y_j,\tilde{Y}_j} [y_{ji}, \tilde{y}_{ji}, \mathcal{T}_{Y_j}](\mathcal{V}_{Y_{jk}})\{y_j\}$$
$$\mathsf{Joint\text{-}Pedersen\text{-}RVSS}(f,l) \xrightarrow[t,n]{Z_j,\tilde{Z}_j} [z_{ji}, \tilde{z}_{ji}, \mathcal{T}_{Z_j}](\mathcal{V}_{Z_{jk}})\{z_j\}$$
$$\mathsf{Joint\text{-}Pedersen\text{-}RVSS}(f,l) \xrightarrow[t,n]{W_j,\tilde{W}_j} [w_{ji}, \tilde{w}_{ji}, \mathcal{T}_{W_j}](\mathcal{V}_{W_{jk}})\{w_j\}$$
$$\mathsf{Joint\text{-}Pedersen\text{-}RVSS}(f,l) \xrightarrow[t,n]{S_j,\tilde{S}_j} [s_{ji}, \ _{ji}, \mathcal{T}_{S_j}](\mathcal{V}_{S_{jk}})\{s_j\}$$
   The servers send their private information about s_j to server $\mathcal{S}_j$.
   (b) The servers perform the following:
   $\mathsf{Shared\text{-}Exp}[\mathcal{T}_{X_j}, \mathcal{T}_{Y_j}, \mathcal{T}_{Z_j}, \mathcal{T}_{W_j}](f, g, h, M_j) \rightarrow (E_j = f^{x_j} g^{y_j} h^{z_j} M_j^{w_j})$
   $\mathsf{Shared\text{-}Exp}[\mathcal{T}_{R_j}](f) \rightarrow (F_j = f^{r_j})$
   $\mathsf{Shared\text{-}Exp}[\mathcal{T}_{R_j}](g) \rightarrow (G_j = g^{r_j})$
   $\mathsf{Shared\text{-}Exp}[\mathcal{T}_{R_j}, \mathcal{T}_{P_j}](h, f) \rightarrow (I_j = h^{r_j} f^{pw})$
   $\mathsf{Abe\text{-}Mult}[\mathcal{T}_{Z_J}, \mathcal{T}_P](f,l) \xrightarrow[t,n]{} [\chi_{ji}, \tilde{\chi}_{ji}, \mathcal{T}_{\chi_j}](\mathcal{V}_{\chi_{jk}}, QUAL)\{\chi_j = z_j \cdot pw\}$
   (c) The servers can now compute
$$\beta_j = \mathcal{H}(\mathcal{S}_j|E_j|F_j|G_j|I_j) \text{ and } N_j = cd^{\beta_j} \bmod p$$
   and
$$\mathsf{Shared\text{-}Exp}[\mathcal{T}_{R_j}](N_j) \rightarrow (J_j = N_j^{r_j})$$
   The messages $\mathcal{S}_j|E_j|F_j|G_j|I_j|J_j$ are sent to the client.
3. For each $j = 1 \dots n$, the client $\mathcal{C}$ runs the next step of the KOY protocol which results in the messages $K_j|\mathsf{Sig}_j$ sent to the servers. The client also computes the corresponding session key sk_j.
4. (a) The servers can verify the signatures and if they are not correct, they stop. Otherwise for each $j = 1 \dots n$ they compute:
   $\mathsf{Shared\text{-}Exp}[\mathcal{T}_{S_j}, \mathcal{T}_{R_j}, \mathcal{T}_{X_j}, \mathcal{T}_{Y_j}, \mathcal{T}_{Z_j}, \mathcal{T}_{W_j}, \mathcal{T}_{\chi_j}](l, K, A, B, C, D, f^{-1}) \rightarrow (mask_j)$
   where $mask_j = l^{s_j} K^{r_j} A^{x_j} B^{y_j} C^{z_j} D^{w_j} f^{-z_j \cdot pw}$
   (b) Server $\mathcal{S}_j$ privately computes the session key as $sk_j = l^{-s_j} \cdot mask_j$.

Fig. 4. Distributed version of the KOY protocol

8. Y. G. Desmedt, Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, July 1994.
9. P. Feldman, A Practical Scheme for Non-Interactive Verifiable Secret Sharing. In 28^{th} *IEEE Symposium on Foundation of Computer Science (FOCS)*, IEEE, pp. 427–437, 1987.
10. W. Ford and B. Kaliski, Server-assisted generation of a strong secret from a password. In 5^{th} *IEEE International Workshop on Enterprise Security*, 2000.
11. Y. Frankel, P. MacKenzie and M. Yung, Adaptively-Secure Distributed Public-Key Systems. In *European Symposium on Algorithms (ESA '99)*, Springer LNCS 1643, pp. 4–27, 1999.
12. P. Gemmell, An Introduction to Threshold Cryptography. *RSA Laboratories' CRYPTOBYTES*, vol. 2, n. 3, Winter 1997.
13. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, The (in)security of distributed key generation in dlog-based cryptosystems. In *EUROCRYPT '99*, Springer LNCS 1592, pp. 295–310, 1999.
14. O. Goldreich and Y. Lindell, Session Key Generation using Human Passwords Only. In *CRYPTO 2001*, Springer LNCS 2139, pp. 408–432, 2001.
15. L. Gong, T. M. A. Lomas, R. M. Needham and J. H. Saltzer, Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, 1993.
16. S. Halevi and H. Krawczyk, Public-key cryptography and password protocols. *ACM Transactions on Information and System Security*, 2(3):230–268, 1999.
17. A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In 4^{th} *ACM Conference on Computers and Communication Security*, ACM Press, pp. 100–110, 1997.
18. A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, Proactive secret sharing, or: How to cope with perpetual leakage. In *CRYPTO '95*, Springer LNCS 963, pp. 339–352, 1995.
19. D. P. Jablon, Strong password-only authenticated key exchange. *ACM Computer Communication Review*, ACM SIGCOMM, 26(5):5–26, October 1996.
20. D. P. Jablon, Password authentication using multiple servers. In *RSA Security Conference 2001*, Springer LNCS 2020, pp. 344–360, 2001.
21. M. Jakobsson, P. MacKenzie and T. Shrimpton, Threshold Password-Authenticated Key Exchange. In *CRYPTO 2002*, Springer LNCS 2442, pp. 385–400, 2002.
22. S. Jarecki and A. Lysyanskaya, Adaptively Secure Threshold Cryptography: Introducing Concurrency, Removing Erasures. In *EUROCRYPT 2000*, Springer LNCS 1807, pp. 221–242, 2000.
23. J. Katz, R. Ostrovsky and M. Yung, Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT 2001*, Springer LNCS 2045, pp. 475–494, 2001.
24. R. Ostrovsky and M. Yung, How to Withstand Mobile Virus Attacks. In 10^{th} *ACM Conference on Principles of Distributed Systems*, ACM Press, pages 51–59, 1991.
25. T. Pedersen, Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '91*, Springer LNCS 576, pp. 129–140, 1991.
26. T. Rabin, A simplified Approach to Threshold and Proactive RSA. In *CRYPTO '98*, Springer LNCS 1462, pp. 89–104, 1998.
27. A. Shamir, How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.