Available online at www.sciencedirect.com

ScienceDirect

Journal of Computer and System Sciences 72 (2006) 978–1001

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

www.elsevier.com/locate/jcss

Provably secure threshold password-authenticated key exchange

Mario Di Raimondo [a,1], Rosario Gennaro [b,*]

[a] *Dipartimento di Matematica e Informatica, Università di Catania, Italy*
[b] *IBM T.J. Watson Research Center, USA*

Received 1 March 2004; received in revised form 29 December 2005

Available online 20 March 2006

Abstract

We present two protocols for threshold password authenticated key exchange. In this model for password authentication, 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.
© 2006 Elsevier Inc. All rights reserved.

Keywords: Authenticated key exchange; Password; Threshold cryptography

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 in the password dictionary without ever interacting with the server until he gets the correct password. Thus a

[*] Corresponding author.
E-mail addresses: diraimondo@dmi.unict.it (M. Di Raimondo), rosario@watson.ibm.com (R. Gennaro).
[1] Work done while visiting IBM Research.

0022-0000/$ – see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcss.2006.02.002

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 [9] 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.[2]

In this paper we present two new provably secure protocols 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 [14,18]. A formal model of security was proposed by Halevi and Krawczyk in [15], 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 [2,5]. The first (and only currently known) protocol to achieve security without *any* additional setup is that of Goldreich and Lindell [13]. 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 [13] 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 [22] 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 [9] 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 [19] improves on [9], 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 [20].

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 [7,11] for surveys of this research area. We use and modify several existing threshold cryptography protocols, most importantly Feldman's and Pedersen's VSS protocols [8,24], their application to discrete-log key generation from [12] and the protocol for multiplication of shared secrets by Abe [1].

1.2. The notion of threshold password-authenticated key exchange

In a *key exchange* (KE) (or *agreement*) protocol, two parties (usually referred to as the *client* and the *server*), run a protocol which terminates with them agreeing on a common random value which is secret to any other party in the

[2] It should be noted that an alternative solution would be to use secure hardware to protect the memory area where the password is stored.

network. Such random value can then be used as a key to encrypt and authenticate the communication in the session. In the basic version of this protocol the adversary is supposed to be passive, i.e., limited to eavesdropping. In other words, the communication channels connecting parties are assumed to be *authentic*, i.e., guarantee that messages received have not been modified.

A more realistic setting is the one of non-authentic channels, where the adversary may modify messages sent between honest parties. In this case an *authenticated key exchange* protocol (AKE) must also guarantee the identity of the two parties taking place. In a *password-authenticated key exchange* (PAKE) protocol, this guarantee is built upon the fact that the two parties share a secret (humanly-memorizable) password.

As we said above, our goal is to protect the password from hackers' attacks on the authenticating server. In order to do so we share the password among n servers, so that at least $t + 1$ of them must be compromised in order to learn the password. This technique, however, raises the following question: if no server in particular knows the password, who is going to play the role of the authenticating server, and who will ultimately share a key with the client? There are two possible answers to this question.

Transparent protocols

The first option is to enforce a *transparency* property on the PAKE protocol: to the eyes of the client the protocol should look exactly like a normal PAKE protocol. The client is not 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. The client interacts with a *gateway* server which to the client's eyes will be the authentication server. At the end the secret key will be shared between the client and the gateway.

The main advantage of this solution is its efficiency and simplicity. 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.

In order to avoid creating single points of failure, we assume that all servers are "equal," i.e., they can all play the gateway role during the computation and thus they will all learn the established secret key. Thus in this setting, the client establishes a *single* key with the servers. This means that the adversary can learn the key by breaking into a single server (e.g., the gateway). Thus the "transparent option" guarantees the following security properties: (i) if the adversary does not break into any server, then the protocol is a secure PAKE; (ii) if the adversary breaks into less than t servers including the gateway, 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.

The definition of security in this case is very simple: it is 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 $\widehat{pw}$ (conditioned to the event that the adversary does not guess the right password, in which case naturally the two views are distinguishable).

Non-transparent protocols

The notion of transparent protocol corresponds to the intuitive notion of preserving the security of the passwords, as long as no more than t servers are compromised. For many applications this could be sufficient. But what if we want to protect the security of the *sessions*?

Consider the example of a Kerberos-like password system. Here the client needs to authenticate itself in order to access several services. He first authenticates itself with a centralized server, who then grants him *tickets* to access the various services that are authorized to the client. This way the client has to perform an expensive authentication only once, and can access the various services quickly. In this scenario a transparent protocol can be quite problematic, since the adversary by breaking into a single server can obtain tickets for *all* the services.

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.[3] 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.

Notice how this immediately improves security in the Kerberos scenario discussed above. In this case to access a service we can require the client to present more than t tickets, one from each authenticating server. Now the adversary must break into more than t servers to access any service.

There is of course a cost to pay for this improved security. Non-transparent protocols are less efficient and more complicated. Also the definition of security in this case is more complex. Basically we need to require that each key agreement between the client and each of the n server is a secure PAKE.

1.3. Our solution in a nutshell

Our starting point was the KOY protocol from [22]. 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 [26] (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 difference between the two protocols is that the first is a transparent one, while the second is non-transparent.

The first protocol

Here we enforce the *transparency* property: to the eyes of the client the protocol should look exactly like a centralized KOY protocol. All the messages exchanged by the client and the gateway will follow the pattern of a regular KOY protocol.

The second protocol

The second protocol is non-transparent: here the client will basically run n copies of the KOY protocol, one with each server. The servers will cooperate to compute together the answers of the ith server in the ith execution of the KOY protocol.

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 $\mathcal{A}_{\text{thresh}}$, 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 then be 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 $\mathcal{A}_{\text{thresh}}$ on this virtual threshold world can be translated in a successful attack in the centralized instance.

Fault-tolerance threshold

Our protocols achieve security against an adversary who corrupts at most $t < n/3$ servers. It is possible to improve the fault-tolerance to $t < n/2$, but the resulting protocols would be a lot more complicated. We decided to focus on simplicity rather than optimal threshold.

[3] 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 wrong parameters n, t 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 [16,17,23] 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. A detailed description appears in Appendix F.

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 $\bmod p$ 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 $1/2$. A formal statement of the DDH Assumption is in Appendix A. For a discussion on the DDH see [4].

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. A typical assumption for password protocols is that the communication between the clients and the servers is totally asynchronous, i.e., messages can be delivered with any delay, and sessions can be interleaved arbitrarily.

On the other hand, we assume that the server's side is implemented via a set of n servers $\mathcal{S}_1, \ldots, \mathcal{S}_n$. 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 between the servers 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.

Notice that there is no contradiction with the fact that we assume asynchronous communication between the clients and the servers, while we assume a partial synchronicity between the servers themselves. Indeed in typical applications the communication between clients and servers will happen on a "low-end" medium, while one expects the distributed servers to be connected via a better dedicated network.

2.1. Password authenticated key exchange

Here we briefly recall the formal security model for centralized password key exchange protocols from [2,5]. We then explain how to extend it for the case of a distributed server.

A protocol for password key exchange assumes that there is a set of *principals* which are the clients and the servers who will engage in the protocol. Each client stores his own private password pw, while each server stores all the passwords of the clients who have access to that server. We assume that the client may use the same password with different servers. Each principal may start various executions of the protocol, with different partners. Thus we denote with $\Pi_i^{l_i}$ the l_ith instance that user P_i runs. Instances maintain state, which is updated as the protocol progresses. In particular the state of an instance $\Pi_i^{l_i}$ includes the following variables (initialized as *null*):

- $\mathsf{sid}_i^{l_i}$ which is the *session identifier* of this particular instance;
- $\mathsf{pid}_i^{l_i}$ which is the *partner identifier*, i.e., the name of the principal with whom $\Pi_i^{l_i}$ believes he's interacting;
- $\mathsf{acc}_i^{l_i}$ which is a boolean variable denoting if $\Pi_i^{l_i}$ accepts or rejects at the end of the execution.

Partnering. We say that two instances $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ are *partnered*, if (1) $\mathsf{sid}_i^{l_i} = \mathsf{sid}_j^{l_j} \neq \mathit{null}$; (2) $\mathsf{pid}_i^{l_i} = j$ and $\mathsf{pid}_j^{l_j} = i$.

The adversary is given total control of the external network (i.e., the network connecting clients to servers). In particular we assume that the adversary can, not only listen to the messages exchanged by players, but also interject messages of her choice and/or stop messages sent by the parties. A way to model such an adversary is to give her oracle access to the instances of the protocol run by the principals. In particular the adversary can make these oracle calls:

- $\mathsf{Execute}(i, l_i, j, l_j)$ – This call forces an execution of the protocol between instances $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ (where P_i is a client and P_j is a server). The output is the transcript of the protocol. This allows the adversary to eavesdrop on a regular execution.
- $\mathsf{Send}(i, l_i, M)$ – This call sends the message M to the instance $\Pi_i^{l_i}$. The output is whatever this instance outputs after receiving the message M (given the current state of the instance $\Pi_i^{l_i}$). This allows the adversary to interject spurious messages in the protocol.
- $\mathsf{Reveal}(i, l_i)$ – This call outputs the secret key $sk_i^{l_i}$ which resulted from instance $\Pi_i^{l_i}$. This allows the adversary to learn past secret key from previous executions, modeling improper exposure of past session keys.
- $\mathsf{Test}(i, l_i)$ – This call is needed to define security. The adversary is allowed to query it only once, and the output is either $sk_i^{l_i}$ or a random key sk (each case happens with probability $1/2$).

We can state a *correctness* condition for the protocol as follows:

Correctness. If two partnered instances $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ accept (i.e., $\mathsf{acc}_i^{l_i} = \mathsf{acc}_j^{l_j} = 1$), then they must end with the same secret key (i.e., $sk_i^{l_i} = sk_j^{l_j}$).

We can now define what it means for a protocol to be secure. Basically we want the secret key sk generated by the protocol to be totally hidden from the adversary. Thus we say that the adversary succeeds if for the Test query she guesses if the output is a random key or the real key. Of course we need to restrict the instances on which the adversary can call the Test query in order to have a meaningful notion (otherwise the adversary could call Test on an instance on which she previously called Reveal). We define the adversary's *advantage* as

$$\mathsf{Adv}(\mathcal{A}) = \left| \mathsf{Prob}[\mathcal{A} \text{ succeeds}] - \frac{1}{2} \right|.$$

We would like the advantage to be negligible, but we know that $\mathcal{A}$ can always guess the password and immediately succeed. Thus we say that our protocol is secure, if guessing the password is the best that the adversary can do. If the password is chosen uniformly[4] from a dictionary Dict, then we require

[4] This uniformity requirement is made for simplicity and it can be easily lifted by adjusting the advantage with the max a priori probability over all the passwords.

$$\mathsf{Adv}(\mathcal{A}) < \frac{Q_{\text{send}}}{|\mathsf{Dict}|} + \mathsf{negl}(n),$$

where n is the security parameter and Q_{send} the number of Send queries the adversary performs (since those queries are the ones which allow her to search the dictionary), provided that $\mathcal{A}$ does not ask $\mathsf{Test}(i, l_i)$, after asking $\mathsf{Reveal}(i, l_i)$ or $\mathsf{Reveal}(j, l_j)$, where $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ are partnered.

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 [6,10,21]).

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 establishes a unique key sk with the collection of 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 $pw, \widehat{pw}$ the two views $view(pw)$ and $view(\widehat{pw})$ can be distinguished with at most probability $Q_{\text{send}}/|\mathsf{Dict}|$.

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 [22]. 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

KOY

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

1. The client $\mathcal{C}$ generates a key pair (VK, 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: $sk_{\mathcal{C}} = E^{r'} F^{x'} G^{y'} (I f^{-pw})^{z'} J^{w'}$.
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: $sk_{\mathcal{S}} = K^r A^x B^y (C f^{-pw})^z D^w$.

Fig. 1. The KOY protocol.

reduce the security of our distributed version to the security of the original KOY protocol. For details the reader is referred to [22].

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 Fig. 1.

2.3. VSS protocols

Here we recall a few existing techniques that we use in our solutions.

Shamir's secret sharing

In Shamir's secret sharing protocol [26], a dealer shares a secret among n servers $\mathcal{S}_1, \ldots, \mathcal{S}_n$ in the following way. Given a number $t < n$, a prime q and a secret $s \in Z_q$, the dealer chooses at random a polynomial $S(\cdot)$ over Z_q of degree t, such that $S(0) = s$. It then secretly transmits to each server $\mathcal{S}_i$ a share $s_i = S(i) \bmod q$. The secret can easily be reconstructed by polynomial interpolation by at least $t+1$ servers. It is not hard to see that a collection of t servers has no information about the secret. Notice also that this protocol does not tolerate a malicious adversary (a bad dealer could give shares that do not lie on a polynomial of degree t and/or bad servers may prevent reconstruction by giving out bad shares).

We will use the following notation to indicate the above protocol:

$$\mathsf{Shamir\text{-}SS}[s] \xrightarrow[t,n]{S} [s_i].$$

Feldman's VSS

Feldman's Verifiable Secret Sharing (VSS) protocol [8], 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*.

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 kth coefficient of $S(\cdot)$. This will allow the players to check that the values s_i really define a secret by checking that

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

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 Eq. (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.

$$\text{Feldman-VSS}[s](g) \xrightarrow[t,n]{S} [s_i](\mathit{Vs}_k).$$

Pedersen's VSS

We now recall a VSS protocol that provides information theoretic secrecy for the shared secret. This is in contrast to Feldman's VSS protocol which leaks the value of g^s. The protocol is due to Pedersen [24].

Pedersen's VSS uses the parameters p, q, g as defined for Feldman's VSS. In addition, it uses an element $h \in Z_p^*$ such that h belongs to the subgroup generated by g 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 $\mathit{Vs}_k = g^{a_k} h^{b_k}$, where a_k (respectively b_k) is the kth coefficient of S (respectively $\tilde{S}$).

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

$$g^{s_i} h^{\tilde{s}_i} = \prod_k (\mathit{Vs}_k)^{i^k} \mod p. \tag{2}$$

As in Feldman's VSS 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 Eq. (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 $\mathit{Vs}_0 = g^s h^{b_0}$ (it can be seen that for any value s' there is exactly one value b_0' such that $\mathit{Vs}_0 = g^{\sigma'} h^{b_0'}$ and thus Vs_0 gives no information on s).

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

$$\text{Pedersen-VSS}[s, \tilde{s}](g, h) \xrightarrow[t,n]{S,\tilde{S}} [s_i, \tilde{s}_i](\mathit{Vs}_k).$$

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 Vs_k for the final secret can be obtained by multiplying the commitments published by each player.

We will later require the following notation:

$$\text{Joint-Pedersen-RVSS}(g, h) \xrightarrow[t,n]{S,\tilde{S}} [s_i, \tilde{s}_i, \mathit{Ts}](\mathit{Vs}_k, \mathit{QUAL})\{s\}$$

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 Ts and QUAL.

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 Ts 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:

$$\text{Abe-Mult}[s_i, \tilde{s}_i, r_i, \tilde{r}_i, \mathcal{V}s_k, \mathcal{V}r_k](g, h) \underset{t,n}{\longrightarrow} [\chi_i, \tilde{\chi}_i, \mathcal{T}\chi](\mathcal{V}\chi_k, QUAL)\{\chi = rs\}$$

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 us 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 [12]: 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.[5]

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 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:

$$\text{Shared-Exp}[\mathcal{T}s_1, \ldots, \mathcal{T}s_m](g_1, \ldots, g_m) \to \left(g_1^{s_1} \cdots g_m^{s_m}\right).$$

The protocol appears in Fig. 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:

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

where a_{ik} (respectively α_{ik}) are the coefficients of the sharing polynomial used by $\mathcal{S}_i$ during the joint generation of s (respectively 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.

[5] In [12] 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 TR, TS

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

$$E_{ik} = g^{\alpha_{ik}} \cdot h^{a_{ik}} \mod p \quad \text{for } k \in [0, \ldots, t]$$

and broadcasts them.

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

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

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 $k \in [0, \ldots, t]$.

4. Now each player can compute

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

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

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 which we show in Appendix B.

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 one of the servers, trusted only at the beginning) 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 [25], though we use it for different purposes).

That is, the password pw is first split in n random values $pw_1 \cdots pw_n$ such that $pw = pw_1 + \cdots + pw_n \mod q$ (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, after that the sharing informations 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 Fig. 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.[6] 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 Fig. 3.

About efficiency

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

Moreover, we observe that the whole of step 3(a), 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 2. *If $n > 3t$ and the DDH Assumption holds, then the protocol* Dist-KOY1 *is a transparently secure distributed password authentication protocol.*

Proof appears in Appendix C.

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.

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

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 $z_j \cdot pw$.

[6] See Appendix D 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: TP: 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

$$\alpha = \mathcal{H}(\mathcal{C}|\mathsf{VK}|A|B|C) \quad \text{and} \quad M = cd^{\alpha} \mod p.$$

3. (a) The servers $\mathcal{S}_i$ perform the following:

$$\text{Joint-Pedersen-RVSS}(f,l) \xrightarrow[t,n]{R,\tilde{R}} [r_i, \tilde{r}_i, TR](VR_k)\{r\},$$

$$\text{Joint-Pedersen-RVSS}(f,l) \xrightarrow[t,n]{X,\tilde{X}} [x_i, \tilde{x}_i, TX](VX_k)\{x\},$$

$$\text{Joint-Pedersen-RVSS}(f,l) \xrightarrow[t,n]{Y,\tilde{Y}} [y_i, \tilde{y}_i, TY](VY_k)\{y\},$$

$$\text{Joint-Pedersen-RVSS}(f,l) \xrightarrow[t,n]{Z,\tilde{Z}} [z_i, \tilde{z}_i, TZ](VZ_k)\{z\},$$

$$\text{Joint-Pedersen-RVSS}(f,l) \xrightarrow[t,n]{W,\tilde{W}} [w_i, \tilde{w}_i, TW](VW_k)\{w\}.$$

We denote by $QUAL$ the intersection of all the qualified sets in the above five protocols.

(b) The servers perform the following:

$$\text{Shared-Exp}[TX, TY, TZ, TW](f, g, h, M) \to \left(E = f^x g^y h^z M^w\right),$$

$$\text{Shared-Exp}[TR](f) \to \left(F = f^r\right),$$

$$\text{Shared-Exp}[TR](g) \to \left(G = g^r\right),$$

$$\text{Shared-Exp}[TR, TP](h, f) \to \left(I = h^r f^{pw}\right),$$

$$\text{Abe-Mult}[z_i, \tilde{z}_i, p_i, \tilde{p}_i, VZ_k, VP_k](f,l) \xrightarrow[t,n]{} [\chi_i, \tilde{\chi}_i, T\chi](V\chi_k, QUAL)\{\chi = z \cdot pw\}.$$

The servers can now compute

$$\beta = \mathcal{H}(\mathcal{S}|E|F|G|I) \text{ and } N = cd^{\beta} \mod p,$$

where $\mathcal{S}$ is the "name" of the collective authentication server.

(c) The servers can now compute:

$$\text{Shared-Exp}[TR](N) \to \left(J = N^r\right).$$

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:

$$\text{Shared-Exp}[TR, TX, TY, TZ, TW, T\chi](K, A, B, C, D, f^{-1}) \to \left(sk_{\mathcal{S}} = K^r A^x B^y C^z D^w f^{-z \cdot pw}\right)$$

to compute the session key to $sk_{\mathcal{S}}$.

Fig. 3. Transparent distributed version of the KOY protocol.

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.

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 $j = 1, \ldots, n$ sent to the gateway.
2. The servers compute

$$\alpha_j = \mathcal{H}(\mathcal{C}|\mathsf{VK}_j|A_j|B_j|C_j) \quad \text{and} \quad M_j = cd^{\alpha_j} \mod p.$$

Then for each $j = 1, \ldots, n$ the following steps are performed:

(a) The servers perform

$$\text{Joint-Pedersen-RVSS}(f,l) \xrightarrow[t,n]{R_j,\tilde{R}_j} [r_{ji}, \tilde{r}_{ji}, TR_j](VR_{jk})\{r_j\},$$

$$\text{Joint-Pedersen-RVSS}(f,l) \xrightarrow[t,n]{X_j,\tilde{X}_j} [x_{ji}, \tilde{x}_{ji}, TX_j](VX_{jk})\{x_j\},$$

$$\text{Joint-Pedersen-RVSS}(f,l) \xrightarrow[t,n]{Y_j,\tilde{Y}_j} [y_{ji}, \tilde{y}_{ji}, TY_j](VY_{jk})\{y_j\},$$

$$\text{Joint-Pedersen-RVSS}(f,l) \xrightarrow[t,n]{Z_j,\tilde{Z}_j} [z_{ji}, \tilde{z}_{ji}, TZ_j](VZ_{jk})\{z_j\},$$

$$\text{Joint-Pedersen-RVSS}(f,l) \xrightarrow[t,n]{W_j,\tilde{W}_j} [w_{ji}, \tilde{w}_{ji}, TW_j](VW_{jk})\{w_j\},$$

$$\text{Joint-Pedersen-RVSS}(f,l) \xrightarrow[t,n]{S_j,\tilde{S}_j} [s_{ji}, \tilde{s}_{ji}, TS_j](VS_{jk})\{s_j\}.$$

The servers send their private information about s_j to server $\mathcal{S}_j$.

(b) The servers perform the following:

$$\text{Shared-Exp}[TX_j, TY_j, TZ_j, TW_j](f,g,h,M_j) \to \left(E_j = f^{x_j} g^{y_j} h^{z_j} M_j^{w_j}\right),$$
$$\text{Shared-Exp}[TR_j](f) \to \left(F_j = f^{r_j}\right),$$
$$\text{Shared-Exp}[TR_j](g) \to \left(G_j = g^{r_j}\right),$$
$$\text{Shared-Exp}[TR_j, TP_j](h,f) \to \left(I_j = h^{r_j} f^{pw}\right),$$
$$\text{Abe-Mult}[z_{ji}, \tilde{z}_{ji}, p_{ji}, \tilde{p}_{ji}, VZ_{jk}, VP_{jk}](f,l) \xrightarrow[t,n]{} [\chi_{ji}, \tilde{\chi}_{ji}, T\chi_j](V\chi_{jk}, QUAL)\{\chi_j = z_j \cdot pw\}.$$

(c) The servers can now compute

$$\beta_j = \mathcal{H}(\mathcal{S}_j|E_j|F_j|G_j|I_j) \quad \text{and} \quad N_j = cd^{\beta_j} \mod p$$

and

$$\text{Shared-Exp}[TR_j](N_j) \to \left(J_j = N_j^{r_j}\right).$$

The messages $\mathcal{S}_j|E_j|F_j|G_j|I_j|J_j$ are sent to the client.

3. For each $j = 1, \ldots, n$, 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 $j = 1, \ldots, n$ they compute:

$$\text{Shared-Exp}[TS_j, TR_j, TX_j, TY_j, TZ_j, TW_j, T\chi_j]\left(l, K, A, B, C, D, f^{-1}\right) \to (mask_j)$$

where $mask_j = l^{s_j} K^{r_j} A^{x_j} B^{y_j} C^{z_j} D^{w_j} f^{-z_j \cdot pw}$.

4.1. 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.

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

Proof appears in Appendix E.

6. A note on efficiency

The protocols are reasonably efficient, particularly for the client. First of all we point out that the difference in cost between the two protocols Dist-KOY1 and Dist-KOY2 is just a factor of n. Indeed, as it can be easily seen by inspecting the descriptions of the protocols, Dist-KOY2 consists of basically n parallel executions of Dist-KOY1 (plus one extra execution of Joint-Pedersen-RVSS to compute the masks).

The other efficiency question is then, how much efficiency we lose compared to the original KOY protocol by using instead its distributed version.

In the case of the client, the answer is easy: Dist-KOY1 is *exactly the same* as the KOY protocol, so there is no efficiency loss for the transparent protocol. As pointed out above, the non-transparent protocol Dist-KOY2 will cost to the client n times as much as the execution of a single KOY protocol.

For the servers the cost is higher, but still reasonable. In the case of the first protocol Dist-KOY1 the cost for each server is $\approx 2n$ times the cost of a single KOY protocol. Again in the case of Dist-KOY2 there is an extra factor of n bringing the total cost $2n^2$ times higher than the KOY protocol (see Appendix G for a detailed accounting).

We should point out that in practical applications the value of n will probably be small (typical application will assume (2 out of 7) or (3 out of 10) solutions), keeping the efficiency penalty of the threshold solution reasonably limited.

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.

Appendix A. The DDH assumption

Let $PRIMES(n)$ be the set of pairs (p,q) where both p,q are prime numbers, $q|p-1$, $|q| = n$ and $|p| = poly(n)$ for some polynomial $poly(\cdot)$. Choose (p,q) at random in $PRIMES(n)$ and g as a random element of order q in Z_p^*. Given an algorithm $\mathcal{A}$ we can define the following probabilities

$$PR_{\mathcal{A},R}(n) = \mathsf{Prob}_{a,b,c\in_R Z_q}\left[\mathcal{A}\left(p,q,g,g^a,g^b,g^c\right) = 1\right],$$
$$PR_{\mathcal{A},DH}(n) = \mathsf{Prob}_{a,b\in_R Z_q}\left[\mathcal{A}\left(p,q,g,g^a,g^b,g^{ab}\right) = 1\right].$$

We say that the Decisional Diffie–Hellman Assumption holds if, for any probabilistic polynomial time algorithm $\mathcal{A}$ and for any polynomial $P(\cdot)$, there exists and index n_P such that for all $n > n_P$ we have

$$\left|PR_{\mathcal{A},R}(n) - PR_{\mathcal{A},DH}(n)\right| < \frac{1}{P(n)}.$$

Appendix B. Proof of security for Shared-Exp

We construct a simulator which runs on the inputs of the corrupted players and is also given the output value $g^r h^s$. The simulator engages in a simulated execution of the protocol with the adversary. The goal is to create a view for the adversary which is indistinguishable from the view she gets in a real execution of the protocol (the *view* is the collection of messages sent and received by the adversary). Notice that the simulator has to do this without knowing r,s (apart from the output value). Which is exactly the reason why this process proves that no further information about r,s is revealed. The detail of the simulation appears in Fig. 5. For simplicity we are assuming that $QUAL = [1,\ldots,n]$ and that the adversary controls the first t players.

Sim-Shared-Exp

Input: $g^r h^s, \{\alpha_{ik}, a_{ik}\}_{i\in[1,\dots,t]}^{k\in[0,\dots,t]}, \{r_{ji}, \tilde{r}_{ji}, s_{ji}, \tilde{s}_{ji}\}_{i\in[1,\dots,t]}^{j\in[1,\dots,n]}, \{VR_{ik}, VS_{ik}\}_{i\in[1,\dots,n]}^{k\in[0,\dots,t]}$

- Simulation of step 1:
  1. The simulator computes:

  $$E_{ik} = g^{\alpha_{ik}} \cdot h^{a_{ik}} \mod p \quad \text{for } k \in [0,\dots,t],\ i \in [1,\dots,t].$$

  2. The simulator chooses at random the values $E_{i0} \in_R G_q$ for $i \in [t+1,\dots,n-1]$, where G_q is the subgroup of order q in Z_p^*. The last E_{n0} is computed as

  $$E_{n0} = \frac{g^r h^s}{\prod_{i=1}^{n-1} E_{i0}} \mod p.$$

  3. The simulator must now compute the values $(E_{i0}, E_{i1}, \dots, E_{it})$ for the good players ($i \in [t+1,\dots,n]$). These values must "look good" to the adversary, i.e., they must satisfy Eq. (3) for the shares of the bad players.
  Recall that $E_{ik} = g^{\alpha_{ik}} \cdot h^{a_{ik}}$ where α_{ik} (respectively a_{ik}) is the kth coefficient of a polynomial of degree t passing through the t points (j, r_{ij}) (respectively (j, s_{ij})) held by the adversary ($j \in [1,\dots,t]$) and the free term $(0, \alpha_{i0})$ (respectively $(0, a_{i0})$). If the simulator knew α_{i0}, a_{i0} he could compute the coefficients by simple interpolation. Recall that interpolation is a simple linear combination, i.e.,

  $$\alpha_{ik} = \sum_{j=0}^{t} \lambda_{jk} r_{ij} \quad \text{and} \quad a_{ik} = \sum_{j=0}^{t} \lambda_{jk} s_{ij}$$

  for the appropriate Lagrangian coefficients λ_{jk}.
  Thus to compute E_{ik}, it is a simple matter to use the above equation in the exponent of g, h, by forcing $E_{i0} = g^{\alpha_{i0}} h^{a_{i0}}$:

  $$E_{ik} = E_{i0}^{\lambda_{0k}} \prod_{j=1}^{t} [g^{r_{ij}} h^{s_{ij}}]^{\lambda_{jk}}.$$

- For steps 2–4 the simulator just follows the protocol since he has enough information to deal with eventual complaints from bad players.

Fig. 5. The simulator for the protocol Shared-Exp.

We can see that the view of the adversary at the end of the simulation is exactly the same as if she had participated in a real protocol. The transcript consists of the values $(E_{i0}, E_{i1}, \dots, E_{it})$ for $i \in [t+1,\dots,n]$; as single elements they are uniformly distributed in G_q, but in the whole they are in the form[7] $g^{k\text{-coeff}(R_i)} h^{k\text{-coeff}(S_i)}$, where R_i and S_i are two polynomials of degree t over G_q such that:

$$\prod_{i=0}^{n} g^{k\text{-coeff}(R_i)} h^{k\text{-coeff}(S_i)} = g^r h^s.$$

The elements of the simulated transcript follow the same distribution of the real view because of the manner they are chosen.

Appendix C. Proof of security for Dist-KOY1

We want to prove that the protocol Dist-KOY1 described in the previous section is *transparently secure*. Recall that this means two things: (i) if the adversary does not break into any server then this is a secure password authentication protocol; (ii) if the adversary breaks into at most t servers then she has no information about the password.

Case (i) is easily argued by seeing that the protocol follows exactly the same pattern as the KOY protocol. Thus if the KOYprotocol is secure, then so it is Dist-KOY1 in the case the adversary does not break into any servers, since in this case the powers of a distributed adversary are the same as in the case of a centralized one.

[7] k-coeff$(\cdot)$ is the kth coefficient of a polynomial.

Case (ii) will be proven via an indistinguishability argument. We are going to show that the views of the adversary when interacting with a system using password pw and a system using password $\widehat{pw}$ can be distinguished with probability at most $\frac{Q_{\mathsf{send}}}{|\mathsf{Dict}|}$, if the DDH Assumption holds.

With regard to the view of the adversary we can concentrate on the public information that the protocol reveals. Beyond that, the adversary sees only t shares on polynomials of degree $t+1$, with free term some secret information. The t shares are information theoretically independent from the secret, so we can ignore them in our argument. Also we can ignore the public information of the relative VSS's since we are only using Pedersen's VSS and that's also information-theoretic.

C.1. Honest client

We first discuss the case in which the client is honest.[8] In this case the view of the adversary $\mathcal{A}$ when the system uses password pw includes the following values (the definition of α, β follows the protocol description): $[A = f^{r'}, B = g^{r'}, C = h^{r'} f^{pw}, D = (cd^{\alpha})^{r'}, K = f^{x'} g^{y'} h^{z'} (cd^{\beta})^{w'}]$ originating from the client, and the values $[F = f^r, G = g^r, I = h^r f^{pw}, J = (cd^{\beta})^r, E = f^x g^y h^z (cd^{\alpha})^w]$ originating from the server, and finally the secret key $sk = K^r A^x B^y (Cf^{-pw})^z D^w$ generated on the servers' side. We denote the distribution of all these values with $view(pw)$.

We want to prove that under the DDH the two distributions $view(pw)$ and $view(\widehat{pw})$ are computationally indistinguishable (here we can actually prove this stronger statement, in the next case we will get the weaker bound of $\frac{Q_{\mathsf{send}}}{|\mathsf{Dict}|}$ on the distinguishing probability).

We do this via a reduction to the semantic security of the ElGamal encryption scheme. Let us assume that we have an adversary $\mathcal{A}$ that can distinguish between $view(pw)$ and $view(\widehat{pw})$. We show how we can distinguish between ElGamal encryptions of f^{pw} and $f^{\widehat{pw}}$.

We are given an instance of the ElGamal encryption scheme, with public parameters p, q, f where f is an element of order q, and public key h. We are also given a ciphertext F, I which is either an encryption of f^{pw} or an encryption of $f^{\widehat{pw}}$ (i.e., $F = f^r$ and $I = h^r f^{pw}$ or $I = h^r f^{\widehat{pw}}$, with $r \in_R Z_q$).

We build a simulated view as follows. We first set the public parameters using f, h and $g = f^{\gamma}$, $c = f^{\delta}$, $d = f^{\epsilon}$, with $\gamma, \delta, \epsilon \in_R Z_q$.

Then we run the adversary $\mathcal{A}$ and we simulate the honest client with the following message: $A = F \cdot f^{\rho}$, $B = F^{\gamma} \cdot g^{\rho}$, $C = h^{\rho} \cdot I$, and $D = F^{\delta + \epsilon\alpha} \cdot (cd^{\alpha})^{\rho}$. Think of $r + \rho = r'$, then we have the exact distribution of a honest client's first message. We also generate a VK.

We then run the servers, letting the adversary control t of them. First we run executions of the various Joint-Pedersen-RVSS protocols to generate values $\hat{r}, x, y, z, w$ ($\hat{r}$ is a dummy value that will not be used in the rest of the simulation). Then we simulate executions of the protocol Shared-Exp in order to generate the following values: F, $G = F^{\gamma}$, I, $J = F^{\delta + \epsilon\beta}$. The computation of $E = f^x g^y h^z (cd^{\alpha})^w$ can be performed following the regular protocol, and not a simulation.

We simulate the second message of the client, by following the regular protocol. I.e., choose $x', y', z', w' \in_R Z_q$ and set $K = f^{x'} g^{y'} h^{z'} (cd^{\beta})^{w'}$. We also attach the appropriate signature Sig.

The last step is the computation of the secret key sk. Notice that $sk = K^r A^x B^y h^{r'z} D^w$. All terms can be computed by the simulator since K^r can be written as $F^{x' + \gamma y' + \delta w' + \epsilon\beta w'} h^{z'}$. So all it has to do is to simulate the last Shared-Exp to hit the value sk.

Notice that if $I = h^r f^{pw}$, these values follow the distribution of $view(pw)$. Otherwise if $I = h^r f^{\widehat{pw}}$ this is the distribution of $view(\widehat{pw})$.

Thus if the adversary can distinguish between $view(pw)$ and $view(\widehat{pw})$ we can break the semantic security of ElGamal, which implies breaking the DDH.

C.2. The adversarial client

We assume now that the client is not honest, i.e., the adversary is trying to impersonate a real client. This case is more complicated than the previous one.[9] We start with some preliminary lemmas.

[8] Looking at the security definition of the Section 2.1, we can consider this case as the one with the oracles Execute, Reveal and Test enabled.

[9] With reference to the security definition of the Section 2.1, in this case we are admitting the use of the Send oracle (the adversary can send spurious message to get the related replies).

We say that a first client message is *correct* for password pw if it was constructed according to the protocol. Specifically a correct first client message $\mathcal{C}, \mathsf{VK}, A, B, C, D$ has the property that letting $\alpha = \mathcal{H}(\mathcal{C}|\mathsf{VK}|A|B|C)$, then

$$\mathrm{Dlog}_f A = \mathrm{Dlog}_g B = \mathrm{Dlog}_h C f^{-pw} = \mathrm{Dlog}_{cd^\alpha} D. \tag{4}$$

The first lemma states that if the servers use password pw, and the first client message is not correct for password pw, then the secret key sk established by the servers is random and uniformly distributed, given all the public information of the protocol (i.e., for the client).

Lemma 4. *If the first client message $\mathcal{C}, \mathsf{VK}, A, B, C, D$ is not correct for password pw, then the distribution of $sk_{\mathcal{S}} = K^r A^x B^y C^z D^w f^{-z \cdot pw}$ conditioned to the rest of the client's view is uniform.*

Proof. Consider the distribution of the value $sk_{\mathcal{S}} K^{-r} = A^x B^y C^z D^w f^{-z \cdot pw}$. Since r is uniquely determined by the rest of the view, it is enough to prove that this value is uniformly distributed.

The values x, y, z, w are constrained only by the value E, which defines a linear equation in them. If we denote with $\mu = \mathrm{Dlog}_f g$, $\nu = \mathrm{Dlog}_f h$ and $\pi = \mathrm{Dlog}_f cd^\alpha$ we have

$$\mathrm{Dlog}_f E = x + \mu y + \nu z + \pi w. \tag{5}$$

Now the value $sk_{\mathcal{S}}$ defines a similar equation in the unknowns x, y, z, w:

$$\mathrm{Dlog}_f sk_{\mathcal{S}} = r_A x + r_B \mu y + r_C \nu z + r_D \pi w, \tag{6}$$

where $r_A = \mathrm{Dlog}_f A$, $r_B = \mathrm{Dlog}_g B$, $r_C = \mathrm{Dlog}_h C f^{-pw}$ and $r_D = \mathrm{Dlog}_{cd^\alpha} D$. However since the first client message is *not* correct for pw, then Eq. (4) is not satisfied, which means that r_A, r_B, r_C, r_D are *not* all equal to each other. Thus Eqs. (5) and (6) are linearly independent.

This implies that given E, any value of $sk_{\mathcal{S}}$ appears with the same probability (since for any value of $sk_{\mathcal{S}}$ we get the same number of solutions in x, y, z, w to the two equations). □

We now try to bound the probability that the adversary can come up with a correct first client message. Notice that such a message is a valid Cramer–Shoup encryption of the password pw (augmented with the verification key VK). Since Cramer–Shoup is non-malleable, we have that all the correct first client messages that the adversary sees are of no use to her in trying to build a *new* correct first client message.

We say that a first client message $\mathcal{C}, \mathsf{VK}, A, B, C, D$ is *new* if the view of the adversary, up to that point, does not contain a first client message that includes the component A, B, C, D.

Lemma 5. *Under the DDH Assumption, the probability that the adversary outputs a new and correct first client message is bounded by $Q/|\mathsf{Dict}| + \mathsf{negl}$ where Q is the number of attempted adversarial logins.*

Proof. The proof works as a reduction to the non-malleability of the Cramer–Shoup encryption scheme. Let us assume that we have an adversary $\mathcal{A}$ that outputs a new and correct first client message with probability $Q/|\mathsf{Dict}| + \mathsf{poly}^{-1}$. We show how to use $\mathcal{A}$ to break the non-malleability of Cramer–Shoup. In particular we show how to use $\mathcal{A}$ to output a new and correct encryption of pw, given several encryptions of pw. Since pw is chosen randomly in a dictionary of size $|\mathsf{Dict}|$, we need to show that we can output such an encryption with probability substantially better than $1/|\mathsf{Dict}|$.

We assume that we can request two kinds of Cramer–Shoup encryptions for pw: a *client* type which include a verification key VK of our choice in the computation of α. And a *server* type which includes a value E of our choice in the computation of α. In [22] it is shown that these are still non-malleable encryptions.

We run $\mathcal{A}$ and every time she requests to see a protocol between an honest client and the servers, we request two Cramer–Shoup encryptions of pw (a client one and a server one) and use them to simulate an execution of the protocol Dist-KOY1. The details of the simulation are similar to the ones for the honest client described above.

That is we use the client encryption we obtained to compute the first message $\mathcal{C}, \mathsf{VK}, A, B, C, D$. Then we simulate the servers' message by simulating the various Shared-Exp to "hit" the server encryption F, G, I, J (as before the computation of E can be carried out normally as in the real protocol).

Then we correctly choose K as in the protocol for the client, and we sign since we know SK (remember that we chose VK).

The servers' secret key is computed as in the honest case above using the fact that we know all the values x', y', z', w' and x, y, z, w.

At the end of this simulation we take all the first client messages that the adversary output. There are at most Q of them. We choose one of them at random and output it. With probability $1/|\mathsf{Dict}| + \mathsf{poly}^{-1}$ this will be a correct encryption of the password pw which is a contradiction, as desired. $\square$

We are now left with the case that the adversary outputs a first client message which is not new. I.e., she is copying the component A, B, C, D from a honest client. Notice however that if she does not copy the verification key VK as well, then the message is not correct (by the properties of the collision resistant function $\mathcal{H}$).[10] But if she copies the whole message, including the verification key VK, she will not be able to sign correctly at the end so no secret key sk will be established.

Lemma 6. *In an execution in which the adversary copies an honest first client message, a secret key is established only with negligible probability.*

The proof of this lemma is a straightforward reduction to the security of the one-time signature scheme, and is therefore omitted.

We are finally able to prove our main goal at this point. We want to prove that $view(pw)$ and $view(\widehat{pw})$ can be distinguished with probability at most $Q/|\mathsf{Dict}| + \mathsf{negl}$. Given the above lemmas, it is enough to show that they are computationally indistinguishable when the first client messages are not correct. Then the bound will follow from Lemmas 5 and 6.

So we limit ourselves to the case that the first client messages are incorrect. Then the simulation follows the same structure of the honest client one, except that we place a random value for the secret key $sk_{\mathcal{S}}$. This is OK because of Lemma 4. Details follow.

We have an adversary $\mathcal{A}$ who only sends out incorrect first client messages and distinguishes between $view(pw)$ and $view(\widehat{pw})$. We use it to break the semantic security of ElGamal by showing that we can distinguish between ElGamal encryptions of f^{pw} and $f^{\widehat{pw}}$.

We are given an instance of the ElGamal encryption scheme, with public parameters p, q, f where f is an element of order q, and public key h. We are also given a ciphertext F, I which is either an encryption of f^{pw} or an encryption of $f^{\widehat{pw}}$ (i.e., $F = f^r$ and $I = h^r f^{pw}$ or $I = h^r f^{\widehat{pw}}$, with $r \in_R Z_q$).

We build a simulated view as follows. We first set the public parameters using f, h and $g = f^{\gamma}$, $c = f^{\delta}$, $d = f^{\epsilon}$, with $\gamma, \delta, \epsilon \in_R Z_q$.

Then we run the adversary $\mathcal{A}$ and get her first message VK, A, B, C, D.

We then run the servers, letting the adversary control t of them. First we run executions of the various Joint-Pedersen-RVSS protocols to generate values $\hat{r}, x, y, z, w$ ($\hat{r}$ is a dummy value that will not be used in the rest of the simulation). Then we simulate executions of the protocol Shared-Exp in order to generate the following values: F, $G = F^{\gamma}$, I, $J = F^{\delta+\epsilon\beta}$. The computation of $E = f^x g^y h^z (cd^{\alpha})^w$ can be performed following the regular protocol, and not a simulation.

We then obtain K from the adversary. If the signature is not correct we stop, otherwise we choose a random sk. And we simulate the last Shared-Exp in order to hit this value.

Notice that if $I = h^r f^{pw}$, these values follow the distribution of $view(pw)$. Otherwise if $I = h^r f^{\widehat{pw}}$ this is the distribution of $view(\widehat{pw})$.

Thus if the adversary can distinguish between $view(pw)$ and $view(\widehat{pw})$ we can break the semantic security of ElGamal, which implies breaking the DDH.

Appendix D. Technical remark about Dist-KOY1

In the Shared-Exp invocation to compute I, we need to take care of a potential discrepancy in the set of qualified players. If some players have been disqualified during the Joint-Pedersen-RVSS in step 3(a) of the Dist-KOY1 protocol,

[10] By the same argument the adversary cannot use "good" server encryptions F, G, I, J, because they do not include a VK and thus will not be correct first messages.

then the set *QUAL* relative to the value r does not contain all the players. But in step 4 of Shared-Exp we assume that both secrets have the same set *QUAL*, which is the intersection of their respective qualified sets. So if apply Shared-Exp as is, we may miss some components of the password. This problem can be easily solved by exposing all the password information of the disqualified players and adjust the result of Shared-Exp accordingly. I.e., we change the final equation of Shared-Exp

$$I = h^r f^{pw} = \prod_{i \in QUAL} E_{i0} \mod p$$

in the follow:

$$I = h^r f^{pw} = \prod_{i \in \{1,\dots,n\} \setminus QUAL} g^{pw_i} \prod_{i \in QUAL} E_{i0} \mod p,$$

where pw_i is the random component of the password associated to $\mathcal{S}_i$, which can be reconstructed in the clear by the honest players.

Appendix E. Proof of security for Dist-KOY2

We need to prove that Dist-KOY2 is a secure distributed password authentication protocol if the original KOY protocol is secure. We prove this by contradiction. We show how a distributed adversary $\mathcal{A}_{\text{thresh}}$ that breaks Dist-KOY2 can be used as a subroutine by an adversary $\mathcal{A}$ which will break the KOY protocol.

Let us recall what it means to break a distributed password authentication protocol. The adversary $\mathcal{A}_{\text{thresh}}$ is allowed to invoke the commands Execute, Send, Reveal. At the end for one of the executions in which she did not ask for Reveal query for the key of, say, server $\mathcal{S}_n$ he will ask a query Test which will return either the real key sk_n or a random key. The adversary $\mathcal{A}_{\text{thresh}}$ wins if she recognizes the right key with probability substantially better than $1/2$.

We assume without loss of generality that $\mathcal{A}_{\text{thresh}}$ breaks into the first t servers $\mathcal{S}_1 \cdots \mathcal{S}_t$.

The centralized adversary $\mathcal{A}$ will run a simulator which will create a virtual distributed world for $\mathcal{A}_{\text{thresh}}$. This simulator will install a fake password $\widehat{pw} \in_R$ Dict.

For each execution of the distributed protocol that $\mathcal{A}_{\text{thresh}}$ wants to see, the centralized adversary $\mathcal{A}$ will request n sessions of the KOY protocol she is trying to break. She will use these n sessions to "embed" them in the distributed protocol, one for each server.

More specifically when $\mathcal{A}_{\text{thresh}}$ invokes an Execute command, $\mathcal{A}$ will invoke n Execute in the centralized world. This will give n transcripts $(msg_{i,1}, msg_{i,2}, msg_{i,3})$ for $i \in [1, \dots, n]$ to $\mathcal{A}$ (one for each server of the virtual distributed world). We can specify the meaning of[11] these messages: $msg_{i,1} = (\mathcal{C}|\mathsf{VK}_i|A_i|B_i|C_i|D_i)$, $msg_{i,2} = (\mathcal{S}_i|E_i|F_i|G_i|I_i|J_i)$ and $msg_{i,2} = (K_i|\mathsf{Sig}_i)$. The simulator will have to create a view in which the ith component will match $msg_{i,1}, msg_{i,2}, msg_{i,3}$. For $msg_{i,1}, msg_{i,3}$ this is not a problem since the simulator controls the client. But for $msg_{i,2}$ we need to show that the transcript of the distributed protocol can be manipulated to hit this specific value, while still producing a view which is indistinguishable from the real one.

Let us see the details (with the steps enumeration related to Fig. 4), the adversary $\mathcal{A}$:

step 1: simulates the first n messages of the client using the messages $msg_{i,1}$;
step 2a: executes in the virtual distributed world the instances of the Joint-Pedersen-RVSS to choose at random $r_i, x_i, y_i, z_i, w_i, s_i$;
step 2b: simulates the executions of the Shared-Exp in order to "hit" the right messages E_i, F_i, G_i, I_i (from the transcript $msg_{i,2}$); the instances of the Abe-Mult are executed using the faked installed password $\widehat{pw}$;
step 2c: follows the real protocol;
step 3: simulates the replies of the client using the messages $msg_{i,3}$;

[11] Look at Fig. 1 for the meaning of the symbols.

step 4a: executes a normal Shared-Exp to compute the value $\widehat{mask_i} = l^{s_i} K^{r_i} A^{x_i} B^{y_i} C^{z_i} D^{w_i} f^{-z_i \cdot \widehat{pw}}$ (at this step we do not need to cheat during this instance of Shared-Exp);

step 4b: computes for each server the faked secret key $\widehat{sk}_i = l^{-s_j} \cdot \widehat{mask_i}$.

A similar argument holds for the Send command: let us go into the details for each type of message that the adversary $\mathcal{A}_{\text{thresh}}$ can send:

1st *message* ($\mathcal{C} \to \mathcal{S}_i$): if $\mathcal{A}_{\text{thresh}}$ sends a message $(\mathcal{C}|\mathsf{VK}_i|A_i|B_i|C_i|D_i)$ then $\mathcal{A}$ forwards the request to the real centralized server, getting a reply: $(\mathcal{S}|E|F|G|I|J)$. This reply will be embedded in the virtual distributed world "hitting" these values as seen in the previous steps;

2nd *message* ($\mathcal{S}_i \to \mathcal{C}$): if $\mathcal{A}_{\text{thresh}}$ sends a message $(\mathcal{S}_i|E_i|F_i|G_i|I_i|J_i)$ to the client, then $\mathcal{A}$ will reply with the answer $(K|\mathsf{Sig})$ which she has received from the client in the real world (relabeled as $(K_i|\mathsf{Sig_i})$);

3rd *message* ($\mathcal{C} \to \mathcal{S}_i$): if $\mathcal{A}_{\text{thresh}}$, impersonating the client, sends a message $(K_i|\mathsf{Sig_i})$ to the server $\mathcal{S}_i$, then $\mathcal{A}$ does not reply anything, but she forwards the message to the real centralized server $\mathcal{S}$ (in the correct instance). Furthermore, $\mathcal{A}$ simulates in the virtual distributed world the computation of the $\widehat{mask_i}$ (from which follows the $\widehat{sk}_i$) using the received K_i and the faked installed password $\widehat{pw}$.

When $\mathcal{A}_{\text{thresh}}$ asks a Reveal on any server, the adversary $\mathcal{A}$ will ask a Reveal on the corresponding centralized session and forward the answer to $\mathcal{A}_{\text{thresh}}$.

Finally we show how to deal with Test. In this case also $\mathcal{A}$ asks her Test query in the centralized case and gets a string which is passed on to $\mathcal{A}_{\text{thresh}}$. Then $\mathcal{A}$ answers whatever $\mathcal{A}_{\text{thresh}}$ answers (i.e., true key or random key).

Clearly, the advantage of $\mathcal{A}$ is exactly the same as the advantage of $\mathcal{A}_{\text{thresh}}$. Indeed the simulated view of $\mathcal{A}_{\text{thresh}}$ is exactly the same as the view of a real execution of the protocol (this is because the simulation of Shared-Exp is perfect).

Appendix F. Proactive security

If a server is compromised, a possible countermeasure is to "reset" the machine (with a "clean copy" of the system software). We assume that this procedure is sufficient to guarantee that the adversary does not control the server anymore.

There is a problem: the share is by now compromised and we do not want the adversary to "collect" enough shares to reconstruct the password. We can prevent this scenario by "refreshing" the shares of the password thanks to the techniques of "proactive security" [16,17,23]. In this way the whole set of servers collaborates in order to randomize the distributed information, keeping the secret unchanged.

This regeneration phase should be periodically executed every τ days, so that the confidential information is secure against an adversary that can break in at most t servers in τ days.

Let us go into the details of the "refreshing" protocol: remember that the password pw is distributed as $pw = \sum_{i \in [1,\dots,n]} a_{i0}$. During the execution of the protocol there will be a server with a special role; we are choosing the server $\mathcal{S}_n$ but this is a merely arbitrary choice. The protocol appears in Figs. 6 and 7.

In this protocol, the first $(n-1)$ servers respectively choose two new random polynomials of degree t to randomize their personal polynomials $P_i(x)$ and $\tilde{P}_i(x)$. Each server sums these new polynomials to the old ones and then it sends new commitments in broadcast. Furthermore it gives each server a point on these randomizer polynomials, so they can update their points in step 2.

If we sum all the new randomizer polynomials we have:

$$\Delta(x) = \hat{P}_1(x) + \cdots + \hat{P}_{n-1}(x), \qquad \tilde{\Delta}(x) = \hat{\tilde{P}}_1(x) + \cdots + \hat{\tilde{P}}_{n-1}(x).$$

These polynomials are not completely known by anyone, but each server $\mathcal{S}_i$ (with $i \in [1,\dots,n-1]$) can compute a point on them (the $\delta_j, \tilde{\delta}_j$ of step 2). These points are sent to the server $\mathcal{S}_n$ so it can reconstruct these two polynomials in step 3. Server $\mathcal{S}_n$ subtracts these polynomials from the personal ones, adjusting them.

Regeneration protocol (1st part)

1. Every server $\mathcal{S}_i$, with $i \in [1, \ldots, n-1]$, chooses two new random polynomials on $Z_q[x]$ of degree t:

$$\hat{P}_i(x) = \hat{a}_{i0} + \hat{a}_{i1}x + \cdots + \hat{a}_{it}x^t, \qquad \hat{\tilde{P}}_i(x) = \hat{b}_{i0} + \hat{b}_{i1}x + \cdots + \hat{b}_{it}x^t.$$

The new personal polynomials of the server $\mathcal{S}_i$ will be:

$$\mathsf{new}P_i(x) = P_i(x) + \hat{P}_i(x) \mod q, \qquad \mathsf{new}\tilde{P}_i(x) = \tilde{P}_i(x) + \hat{\tilde{P}}_i(x) \mod q$$

so the new coefficients will be respectively:

$$\mathsf{new}a_{ik} = a_{ik} + \hat{a}_{ik} \mod q, \qquad \mathsf{new}b_{ik} = b_{ik} + \hat{b}_{ik} \mod q \quad k \in [0, \ldots, t].$$

The server $\mathcal{S}_i$ sends in broadcast the new commitments:

$$\mathsf{new}VP_{ik} = f^{a_{ik}} l^{b_{ik}} \mod p \quad k \in [0, \ldots, t].$$

The server $\mathcal{S}_i$ sends to the server $\mathcal{S}_j$, with $j \in [1, \ldots, n]$, the values:

$$\hat{p}_{ij} = \hat{P}_i(j), \qquad \hat{\tilde{p}}_{ij} = \hat{\tilde{P}}_i(j)$$

2. Every server $\mathcal{S}_j$, with $j \in [1, \ldots, n-1]$, computes the new points on the polynomials of the other servers (using also the old informations):

$$\mathsf{new}p_{ij} = p_{ij} + \hat{p}_{ij} \mod q, \qquad \mathsf{new}\tilde{p}_{ij} = \tilde{p}_{ij} + \hat{\tilde{p}}_{ij} \mod q \quad i \in [1, \ldots, n-1].$$

The server $\mathcal{S}_j$ sends to the server $\mathcal{S}_n$ the values:

$$\delta_j = \sum_{i=1}^{n-1} \hat{p}_{ij} \mod q, \qquad \tilde{\delta}_j = \sum_{i=1}^{n-1} \hat{\tilde{p}}_{ij} \mod q$$

which are, respectively, points on the polynomials:

$$\Delta(x) = \hat{P}_1(x) + \cdots + \hat{P}_{n-1}(x), \qquad \tilde{\Delta}(x) = \hat{\tilde{P}}_1(x) + \cdots + \hat{\tilde{P}}_{n-1}(x).$$

The server $\mathcal{S}_j$ computes the new points on the polynomials of the server $\mathcal{S}_n$, which will be:

$$\mathsf{new}p_{nj} = p_{nj} - \delta_j \mod p, \qquad \mathsf{new}\tilde{p}_{nj} = \tilde{p}_{nj} - \tilde{\delta}_j \mod p.$$

Fig. 6. Protocol to refresh the shares of the password.

Regeneration protocol (2nd part)

3. The server $\mathcal{S}_n$ received the points $(\delta_j, \tilde{\delta}_j)$ with $j \in [1, \ldots, n-1]$ and now it can computes its points $(\delta_n, \tilde{\delta}_n)$. Using $t+1$ points, it can interpolate the polynomials $\Delta(x), \tilde{\Delta}(x)$. The two new personal polynomials of the server $\mathcal{S}_n$ will be:

$$\mathsf{new}P_n(x) = P_n(x) - \Delta(x) \mod q, \qquad \mathsf{new}\tilde{P}_n(x) = \tilde{P}_n(x) - \tilde{\Delta}(x) \mod q$$

so the new coefficients will be respectively (with $k \in [0, \ldots, t]$):

$$\mathsf{new}a_{nk} = a_{nk} - \sum_{i=1}^{n-1} \hat{a}_{ik} \mod q, \qquad \mathsf{new}b_{nk} = b_{nk} - \sum_{i=1}^{n-1} \hat{b}_{ik} \mod q.$$

The server $\mathcal{S}_n$ sends in broadcast the new commitments:

$$\mathsf{new}VP_{nk} = f^{a_{nk}} l^{b_{nk}} \mod p \quad k \in [0, \ldots, t].$$

The servers $\mathcal{S}_n$ computes the new points on the polynomials of the other servers:

$$\mathsf{new}p_{in} = p_{in} + \hat{p}_{in} \mod q, \qquad \mathsf{new}\tilde{p}_{in} = \tilde{p}_{in} + \hat{\tilde{p}}_{in} \mod q \quad i \in [1, \ldots, n-1].$$

4. Every server destroy any information about the old shares.

Fig. 7. Protocol to refresh the shares of the password.

Before these operations, the sum of all the personal polynomials, whose free term is the password pw, was:

$$P(x) = \sum_{i=1}^{n} P_i(x) \mod q$$

now it is:

$$\begin{aligned}\mathsf{new}P(x) &= \sum_{i=1}^{n}\big(\mathsf{new}P_i(x)\big) = \sum_{i=1}^{n-1}\big(P_i(x)+\hat{P}_i(x)\big) + \big(P_n(x) - \Delta(x)\big)\\ &= \sum_{i=1}^{n-1} P_i(x) + \sum_{i=1}^{n-1}\hat{P}_i(x) + \left(P_n(x) - \sum_{i=1}^{n-1}\hat{P}_i(x)\right) = \sum_{i=1}^{n} P_i(x) = P(x) \mod q.\end{aligned}$$

So all the personal polynomials have been randomized, but their sum is invariant as the interpolated secret pw.

Appendix G. Efficiency analysis

An easy inspection of the KOY protocol shows that the cost for a server is about 15 exponentiation (without considering the cost of exponentiating to pw since that's usually a small value).

The cost of Dist-KOY1 for a server is to perform: 5 Joint-Pedersen-RVSS protocols; 5 Shared-Exp protocols (one of them on 4 secrets, one on 2 secret and the rest on 1 secret); and 1 execution of Abe-Mult.

The cost of Joint-Pedersen-RVSS for a player is $\approx 2(n+t)$ exponentiations. Similarly the cost of Shared-Exp on ℓ secrets is $\approx \ell n$ exponentiations. Finally the main cost of Abe-Mult is for each player to perform three Pedersen-VSS protocol and verify the $3(n-1)$ Pedersen-VSS protocols of the remaining $n-1$ players: the cost is thus for each player the same as performing three Joint-Pedersen-RVSS.

Thus a rough estimate of the server's cost of Dist-KOY1 is about $16(n+t)$ exponentiations (from the eight Joint-Pedersen-RVSS) and $9n$ exponentiation (from the various Shared-Exp). Considering that $t = n/3$ the total is less than $31n$, i.e., $\approx 2n$ the cost of a single KOY protocol.

References

[1] M. Abe, Robust distributed multiplication without interaction, in: Advances in Cryptology—Proceedings of CRYPTO '99, in: Lecture Notes in Comput. Sci., vol. 1666, Springer-Verlag, 1999, pp. 130–147.
[2] M. Bellare, D. Pointcheval, P. Rogaway, Authenticated key exchange secure against dictionary attacks, in: Advances in Cryptology—Proceedings of EUROCRYPT 2000, in: Lecture Notes in Comput. Sci., vol. 1807, Springer-Verlag, 2000, pp. 139–155.
[3] S.M. Bellovin, M. Merritt, Encrypted key exchange: Password based protocols secure against dictionary attacks, in: Proceedings 1992 IEEE Symposium on Research in Security and Privacy, IEEE Computer Society Press, 1992, pp. 72–84.
[4] D. Boneh, The decision Diffie–Hellman problem, in: Algorithmic Number Theory—Proceedings of Third Algorithmic Number Theory Symposium, in: Lecture Notes in Comput. Sci., vol. 1423, Springer-Verlag, 1998, pp. 48–63.
[5] V. Boyko, P.D. MacKenzie, S. Patel, Provably secure password-authenticated key exchange using Diffie–Hellman, in: Advances in Cryptology—Proceedings of EUROCRYPT 2000, in: Lecture Notes in Comput. Sci., vol. 1807, Springer-Verlag, 2000, pp. 156–171.
[6] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, Adaptive security for threshold cryptosystems, in: Advances in Cryptology—Proceedings of CRYPTO '99, in: Lecture Notes in Comput. Sci., vol. 1666, Springer-Verlag, 1999, pp. 98–115.
[7] Y.G. Desmedt, Threshold cryptography, Eur. Trans. Telecommun. 5 (4) (July 1994) 449–457.
[8] P. Feldman, A practical scheme for non-interactive verifiable secret sharing, in: Proceedings of the 28th IEEE Symposium on Foundation of Computer Science (FOCS), IEEE, 1987, pp. 427–437.
[9] W. Ford, B. Kaliski, Server-assisted generation of a strong secret from a password, in: Proceedings of the 5th IEEE International Workshop on Enterprise Security, 2000.
[10] Y. Frankel, P. MacKenzie, M. Yung, Adaptively-secure distributed public-key systems, in: Algorithms—Proceedings of European Symposium on Algorithms, ESA '99, in: Lecture Notes in Comput. Sci., vol. 1643, Springer-Verlag, 1999, pp. 4–27.
[11] P. Gemmell, An introduction to threshold cryptography, in: RSA Laboratories' CRYPTOBYTES, vol. 2, No. 3, Winter 1997.
[12] R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, The (in)security of distributed key generation in dlog-based cryptosystems, in: Advances in Cryptology—Proceedings of EUROCRYPT '99, in: Lecture Notes in Comput. Sci., vol. 1592, Springer-Verlag, 1999, pp. 295–310.
[13] O. Goldreich, Y. Lindell, Session key generation using human passwords only, in: Advances in Cryptology—Proceedings of CRYPTO 2001, in: Lecture Notes in Comput. Sci., vol. 2139, Springer-Verlag, 2001, pp. 408–432.
[14] L. Gong, T.M.A. Lomas, R.M. Needham, J.H. Saltzer, Protecting poorly chosen secrets from guessing attacks, IEEE J. Sel. Areas Comm. 11 (5) (1993) 648–656.
[15] S. Halevi, H. Krawczyk, Public-key cryptography and password protocols, ACM Trans. Inf. Syst. Secur. 2 (3) (1999) 230–268.
[16] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, M. Yung, Proactive public key and signature systems, in: Proceedings of the 4th ACM Conference on Computers and Communication Security, ACM Press, 1997, pp. 100–110.
[17] A. Herzberg, S. Jarecki, H. Krawczyk, M. Yung, Proactive secret sharing, or: How to cope with perpetual leakage, in: Advances in Cryptology—Proceedings of CRYPTO '95, in: Lecture Notes in Comput. Sci., vol. 963, Springer-Verlag, 1995, pp. 339–352.
[18] D.P. Jablon, Strong password-only authenticated key exchange, Comput. Comm. Rev. (ACM SIGCOMM) 26 (5) (October 1996) 5–26.

[19] D.P. Jablon, Password authentication using multiple servers, in: Topics in Cryptology—Proceedings of RSA Security Conference 2001, in: Lecture Notes in Comput. Sci., vol. 2020, Springer-Verlag, 2001, pp. 344–360.
[20] M. Jakobsson, P. MacKenzie, T. Shrimpton, Threshold password-authenticated key exchange, in: Advances in Cryptology—Proceedings of CRYPTO 2002, in: Lecture Notes in Comput. Sci., vol. 2442, Springer-Verlag, 2002, pp. 385–400.
[21] S. Jarecki, A. Lysyanskaya, Adaptively secure threshold cryptography: Introducing concurrency, removing erasures, in: Advances in Cryptology—Proceedings of EUROCRYPT 2000, in: Lecture Notes in Comput. Sci., vol. 1807, Springer-Verlag, 2000, pp. 221–242.
[22] J. Katz, R. Ostrovsky, M. Yung, Efficient password-authenticated key exchange using human-memorable passwords, in: Advances in Cryptology—Proceedings of EUROCRYPT 2001, in: Lecture Notes in Comput. Sci., vol. 2045, Springer-Verlag, 2001, pp. 475–494.
[23] R. Ostrovsky, M. Yung, How to withstand mobile virus attacks, in: Proceedings of the 10th ACM Conference on Principles of Distributed Systems, ACM Press, 1991, pp. 51–59.
[24] T. Pedersen, Non-interactive and information-theoretic secure verifiable secret sharing, in: Advances in Cryptology—Proceedings of CRYPTO '91, in: Lecture Notes in Comput. Sci., vol. 576, Springer-Verlag, 1991, pp. 129–140.
[25] T. Rabin, A simplified approach to threshold and proactive RSA, in: Advances in Cryptology—Proceedings of CRYPTO '98, in: Lecture Notes in Comput. Sci., vol. 1462, Springer-Verlag, 1998, pp. 89–104.
[26] A. Shamir, How to share a secret, Commun. ACM 22 (11) (1979) 612–613.