Password-Protected Secret Sharing

Ali Bagherzandi
Dept. of Computer Science
University of California, Irvine
zandi@ics.uci.edu

Stanisław Jarecki
Dept. of Computer Science
University of California, Irvine
stasio@ics.uci.edu

Nitesh Saxena
Computer and Information Sciences
University of Alabama, Birmingham
saxena@cis.uab.edu

Yanbin Lu
Dept. of Computer Science
University of California, Irvine
yanbinl@ics.uci.edu

ABSTRACT

We revisit the problem of protecting user's private data against adversarial compromise of user's device(s) which store this data. We formalize the solution we propose as *Password-Protected Secret-Sharing* (PPSS), which allows a user to secret-share her data among n trustees in such a way that (1) the user can retrieve the shared secret upon entering a correct password into a reconstruction protocol, which succeeds as long as at least $t+1$ uncorrupted trustees are accessible, and (2) the shared data remains secret even if the adversary which corrupts t trustees, with the level of protection expected of password-authentication, i.e. the probability that the adversary learns anything useful about the secret is at most $q/|D|$ where q is the number of reconstruction protocol the adversary manages to trigger and $|D|$ is the size of the password dictionary.

We propose an efficient PPSS protocol in the PKI model, secure under the DDH assumption, using non-interactive zero-knowledge proofs with efficient instantiations in the Random Oracle Model. Our protocol is practical, with fewer than 16 exponentiations per trustee and $8t + 17$ exponentiations per user, with $O(1)$ bandwidth between the user and each trustee, and only three message flows, implying a single round of interaction in the on-line phase. As a side benefit our PPSS protocol yields a new Threshold Password Authenticated Key Exchange (T-PAKE) protocol in the PKI model with significantly lower message, communication, and server computation complexities than existing T-PAKE's.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Cryptographic Controls; D.4.6 [**Security and Protection**]: Authentication

General Terms

Security, Algorithms

CCS'11, October 17–21, 2011, Chicago, Illinois, USA.

Keywords

Secret Sharing, Intrusion Tolerance, Password Authentication

1. INTRODUCTION

A user of computing technology, let's call her Alice, needs to store her valuable private data, e.g. texts, images, passwords, or keys, on her computer or any other electronic device. Alice's device, however, can fall prey to viruses or any other type of malware which could expose her data. Moreover, Alice's device could get lost or stolen and thus fall under control of someone whom Alice does not trust. On the other hand, Alice might want to have access to her private data on more than one device, and she would like her data protected from loss due to device failures.

Consider the general problem of protecting Alice's secret data, let's call it s, in the event of the compromise or failure of the device(s) on which s could be stored. Storing an encryption of s using a key derived from Alice's password is vulnerable to an off-line password dictionary attack once the device is corrupted, and it is also not robust to device failure. Without specialized hardware, Alice could instead outsource s to some trusted remote server, and authenticate to this server whenever she needs to retrieve s. Note that Alice cannot store any authentication token on her device, or otherwise the adversary who corrupts the device can still recover s by using this token to authenticate to the server. If we want a software-only solution, Alice could only use a human-memorable password for authentication to the server. Still, she would be placing all her trust in a single server.

Password-Protected Secret Sharing. To eliminate the bottleneck of a single trusted server we could secret-share s among a set of n trustees so that only a compromise of more than some threshold of trustees would disable the system or leak s. However, secret sharing by itself is not enough because Alice still needs some way to authenticate to these trustees to trigger the secret reconstruction protocol, and since she can only use human-memorable passwords, a black-box combination of password authentication and secret-sharing leaves us with two bad choices: Alice can use n independently chosen passwords, one for each trustee, which is impractical, or she can use the same password with all trustees, but this would eliminate all the benefits of secret-sharing, because a single corrupt trustee could recover this password via an off-line dictionary attack and use it to recover s by authenticating to the other trustees. This motivates the question of how to distribute Alice's data among n trustees so that: (1) Alice can retrieve her data by triggering a reconstruction protocol using only her password,

and the reconstruction is guaranteed as long as at least $t+1$ honest trustees are available; and (2) Alice's data remains secret even when t trustees are corrupted, and the level of this protection is as expected of password-authenticated protocol, i.e. if Alice chooses her password uniformly in set D then the probability that adversary learns anything about Alice's data after triggering q instances of the reconstruction protocol is at most $q/|D|$ plus some negligible quantity. We call a protocol that satisfies these two properties a (t, n) *Password-Protected Secret Sharing (PPSS)*.

Two Implementation Settings. There are two basic implementation settings for a PPSS scheme. In the *roaming user setting* implicitly assumed above, Alice's device does not have any private storage, and the n trustees are implemented by n separate physical entities. This setting is highly robust, because the adversary can destroy Alice's data only by corrupting $n-t$ trustees, and it is easy to extend to multiple user devices, since a user device needs no private information. As for security, the loss of Alice's device(s) gives no information to the adversary, and Alice's data is secure even if the adversary corrupts t out of n trustees. However, Alice's data is vulnerable to on-line dictionary attacks against the trustees, and the adversary corrupting $t+1$ trustees recovers both Alice's data and her password, possibly after an off-line dictionary attack.

In a *private storage setting*, Alice can trade robustness for increased security by initializing an $(n, 2n-t)$ PPSS scheme, distributing n shares among the n trustees, and storing the remaining $n-t$ shares locally on her device. During reconstruction this device will use its shares to play the part of $n-t$ virtual trustees. The resulting system is less robust because Alice's data is lost after corruption of either $n-t$ trustees *or* of Alice's device. However, the system is more secure because an adversary learns Alice's data only after corrupting both $t+1$ trustees *and* Alice's device. Moreover, a network attacker can no longer stage an on-line dictionary attack, because the $n-t$ virtual trustees implemented by Alice's device do not need to respond to network requests.

Applications. A PPSS scheme can be implemented by a dedicated service provider as service to individual users, or it can be administered by users themselves, exploiting their real-world trust relationships, e.g. from on-line social networks such as *Facebook*. For example, using the PPSS scheme in a private storage setting Alice can improve the security of her data compared compared to local storage, at the price of risking data loss if $n-t$ out of n of her trustees become unavailable. The role of the trustees can also be played by different devices belonging to Alice herself. For example, Alice can use PPSS to off-load sensitive data from her mobile phone so that the adversary can gain control of this data only by stealing the phone together with t out of n of her additional devices, e.g. her home computer, her work computer, or even her Bluetooth-connected watch. Various types of trustees could be mixed, so the pool of Alice's own n devices can extend to computers of Alice's Facebook friends and to dedicated service providers. Moreover, each logical trustee can be assigned a different weight by playing the role of $k>1$ virtual trustees. In the PPSS protocol we propose this would increase trustee's computational cost only by replacing one exponentiation with one multi-exponentiation on k bases.

A PPSS scheme can be used to protect any sensitive data, including user's cryptographic keys, e.g. for decryption, signing, authentication, etc. One variant of such application is a *password management*, where Alice's data protected by a PPSS scheme are her passwords to various on-line services. Apart of convenience of having to remember only one "master password", used by the PPSS to recover all the other passwords, such scheme can improve security of password authentication because it allows Alice to use independent random keys instead of all her passwords except the master password. Currently several entities offer such password management service, e.g. LastPass [17] and Mozilla Weave Sync [20], but these are centralized solutions.

Scheme	MSJ06 [19]	RG06 [7]	Ours
Client Computation	$O(1)$	$O(1)$	$O(n)$
Server Computation	$O(n)$	$O(n^2)$	$O(1)$
Total Bandwidth	$O(n^2)$	$O(n^3)$	$O(n)$
Client/Server Messages	7	$\geq$12	3

Figure 1: Previous TPAKE's versus our PPSS/TPAKE.

Related Work. The idea of password-authenticated recovery of secret-shared data was to the best of our knowledge proposed by Ford and Kaliski [11]. As pointed out by a subsequent paper by Jablon [14], the Ford-Kaliski protocol implicitly assumes a *public key infrastructure* (PKI) model, while the improvement given by [14] removed the need for trusted public keys, but both protocol handles only the $t = n$ case of secret-sharing, and neither paper clearly specified the security properties of such scheme nor did they formally argue the security of the protocols they proposed.

The notion of PPSS is closely related to Threshold Password Authenticated Key Exchange (T-PAKE) defined by MacKenzie et al. [19], who followed up on the work of [11, 14], in particular by defining a more general primitive than password-authenticated recovery of secret-shared data, namely a password-authenticated key exchange between a client and n servers. Indeed, any T-PAKE scheme can be used to implement a PPSS scheme at negligible extra cost: The client authenticates to the n trustees with a T-PAKE instance, and the trustees use the keys established by this instance to encrypt and authenticate their shares of Alice's secret. As we show in this paper, in the PKI model the implication works also in the opposite direction, i.e. a PPSS implies a T-PAKE with little extra costs: Alice picks a public key encryption key pair and shares the decryption key using a PPSS scheme. A T-PAKE protocol consists of a PPSS reconstruction followed by each server encrypting a fresh session key under Alice's public key. Alice decrypts these keys using the decryption key recovered in the PPSS instance. For server-to-client authentication each server signs its encrypted session key. In spite of this equivalence between PPSS and T-PAKE in the PKI model, it is productive to formally introduce PPSS as a separate notion, firstly because it is a natural functionality, a protocol that reconstructs a (long) secret if and only if supplied with a correct (short) password, but also because it is simpler than T-PAKE, since only the client has any output, and this output can be dependent across protocol sessions.

Specifically, MacKenzie et al. [19] showed a T-PAKE protocol in the PKI model secure under the Decisional Diffie-Hellman (DDH) assumption in the Random Oracle Model (ROM), while Di Raimondo and Gennaro [7] showed a T-PAKE without relying on ROM and trusted public keys, based on the PAKE of Katz-Ostrovsky-Yung [16]. The drawback of both T-PAKE's is several rounds of server-to-server communication, which is feasible if the servers communicate over a LAN network, as could be the case in fault-tolerant distribution of an authentication server which motivated the T-PAKE's of [19, 7], but the PPSS applications that motivate our work are user-centric and can involve highly heterogeneous trustees, in which case it would be preferable that each trustee communicates only with the user device. In such user-centric communication model, the T-PAKE of [19] requires exchange of 9 messages between the user and each trustee, with $O(n)$ bandwidth, and the costs of T-PAKE of [7] are even higher (see Table 1 below). For the special case of 2 servers these results were improved upon by Brainard et al. [3] in the PKI model, and by Katz et al. [15] in the model without trusted public keys, but even these

2-server T-PAKE's require, respectively, 7 and 5 messages in the user-centric communication model, assuming ROM.

MacKenzie and Reiter's "Capture-Resilient Device" protocols [18] considered essentially the same goals as a PPSS scheme cast in the private storage setting, but in a less general setting where the trust threshold is fixed at $t = 1$, i.e. where a single trustee, in addition to the client's device, is needed to reconstruct user's data. Their schemes considered not only password-protected storage of user's data, but also password-protected computation of RSA signatures and ElGamal decryption. Note that any T-PAKE protocol, including one implied by a PPSS scheme, can be used to implement such password-protected computation given a non-interactive threshold protocol for the respective computation. The servers simply encrypt the messages of the threshold function-computation protocol under the session keys generated by a T-PAKE instance. Thus the 2-party password-protected signature/decryption computation of [18] can be generalized to any (t, n) threshold using our PPSS and non-interactive threshold RSA signature of [24] and non-interactive threshold ElGamal decryption of [6]. (We note, however, that the protocols of [18] for the $t = 1$ case use only 2 messages, while those built using our PPSS scheme would take 3 messages.) Xu and Sandhu [26] consider the same notion of password-authenticated threshold signature computation called TPAKE-TSig, except that they consider it in the roaming user setting. Since their construction is exactly the composition of a T-PAKE and a threshold signature sketched above, their TPAKE-Sig protocol can only be as fast as a T-PAKE.

PPSS is remotely related to the "key-insulated" and "intrusion-resilient" cryptosystems introduced in [9, 8], which also concern protecting user's data against device corruption via secret-sharing user's private key among semi-trusted entities. However, they consider only embedded entities like smartcards or co-processors, to whom the user does not need to authenticate remotely.

Our Contributions. We present a practical PPSS scheme in the PKI model, where the client trusts some public key(s), provably secure under the DDH assumption, assuming non-interactive zero-knowledge proofs which can be efficiently instantiated in ROM. Our scheme requires fewer than 16 exponentiations per trustee and $8t + 17$ exponentiations per user, with $O(1)$ bandwidth between the user and each trustee, and only three message flows, implying a single round of interaction if trustees' first messages are precomputed. We also show that a PPSS implies a T-PAKE with very little computation and communication overhead, and no additional communication rounds. Thus our scheme implies a T-PAKE with fewer communication rounds, less bandwidth, and less server computation than existing T-PAKE's of [19, 7]. The exact cost comparison in the user-mediated communication model is shown in Table 1. For the special case of $n = 2$, our scheme has lower message complexity and server computation costs than even the 2-party T-PAKE's of Brainard et al. [3] and Katz et al. [15]. However, in the private storage setting and threshold t set at 1, our scheme is beaten by those of [18].

2. PPSS DEFINITION

A Password-Protected Secret Sharing (PPSS) scheme is a protocol involving $n + 1$ parties, a user U, and n servers $\mathsf{P}_1, ..., \mathsf{P}_n$. A PPSS scheme for secret space S and dictionary D is a tuple $(\mathsf{Init}, \mathsf{User}, \mathsf{Server})$, where $\mathsf{Init}(p, s)$ is an initialization algorithm which on inputs a secret $s \in S$ and password $p \in D$ generates $\mathsf{st} = (\mathsf{st}_0, \mathsf{st}_1, ..., \mathsf{st}_n)$ where st_0 are public parameters and st_i is the private state of server P_i; $\mathsf{User}(\tilde{p}, \mathsf{st}_0)$ is an interactive algorithm followed by U on its password $\tilde{p}$ (presumably equal to p) and parameters st_0; and $\mathsf{Server}(\mathsf{st}_i)$ is an interactive algorithm followed by P_i on input st_i. The Server algorithm has no local output, while the User algorithm has a local output which is either some bitstring s' or a special rejection symbol $\perp$. We require a PPSS protocol to be complete in the sense that if $\mathsf{PPSS}(p, \mathsf{st})$ is a random variable defined as the local output of algorithm $\mathsf{User}(p, \mathsf{st}_0)$ after an interaction with oracles $\mathsf{Server}(\mathsf{st}_1)$, ..., $\mathsf{Server}(\mathsf{st}_n)$ then $\mathsf{PPSS}(p, \mathsf{Init}(p, s)) = s$ for any $(s, p) \in S \times D$. To model concurrent execution of several PPSS protocol instances we denote by $\mathsf{User}^\diamond(p, \mathsf{st}_0)$ an oracle which allows the caller to interact with any number of $\mathsf{User}(p, \mathsf{st}_0)$ instances. Importantly, the caller sees only protocol messages output by each User instance it interacts with, and not the local output of any of these instances, although this view is crucially amended in the *strong security* notion below. Similarly, for any set B we denote by $\mathsf{Server}^\diamond(\mathsf{st}_{\overline{\mathsf{B}}})$ an oracle which allows the caller to interact with any number of $\mathsf{Server}(\mathsf{st}_i)$ instances for any i in $\overline{\mathsf{B}} \triangleq \{1, ..., n\} \setminus \mathsf{B}$. We say that probabilistic algorithm $\mathcal{A}$ interacts with at most q_U user sessions if in any of its executions $\mathcal{A}$ initializes at most q_U instances of $\mathsf{User}(p, \mathsf{st}_0)$ algorithm when interacting with oracle $\mathsf{User}^\diamond(p, \mathsf{st}_0)$, and we say that $\mathcal{A}$ interacts with at most q_S server sessions if in any execution of $\mathcal{A}$ we have $\sum_{i \in \overline{\mathsf{B}}} q_i \leq q_S$ where q_i is the number of $\mathsf{Server}(st_i)$ instances $\mathcal{A}$ initializes when interacting with oracle $\mathsf{Server}^\diamond(\mathsf{st}_{\overline{\mathsf{B}}})$.

We define security of a PPSS scheme in terms of advantage in distinguishing between PPSS instances initialized with two different secrets, where the adversary sees the public parameters st_0, the private states $\mathsf{st}_\mathsf{B} \triangleq \{\mathsf{st}_i\}_{i \in \mathsf{B}}$ of corrupted servers $\{P_i\}_{i \in \mathsf{B}}$, and has concurrent oracle access to instances of the user and server algorithms executing on inputs defined by the initialization procedure. Note that this defines a PPSS scheme in the PKI model, because the user algorithm is assumed to execute only on the st_0 parameters generated by $\mathsf{Init}(p, s)$, and this st_0 could be certified under a trusted public key instead of being stored locally. We call an (t, n)-threshold PPSS scheme secure if this advantage is bounded by $\frac{1}{|D|}$, the probability of guessing the password, times $\lfloor \frac{q_S}{t - t' + 1} \rfloor$ where $t' \leq t$ is the number of servers an adversary corrupts, plus at most a negligible amount. The last factor corresponds to the probability of success of an on-line dictionary attack in a threshold setting: An adversary who learns the shares of $t' \leq t$ servers can test any password $\tilde{p}$ in D by executing $\mathsf{User}(\tilde{p}, st_0)$ and interacting with any subset of $t - t' + 1$ uncorrupted servers. To the best of our knowledge all previous works on T-PAKE's bound attackers' success by $\frac{q_S}{|D|}$, which is higher than the above bound, except if the adversary corrupts the maximum threshold of servers, $t' = t$. In particular, if each server locally bounds the number of sessions it performs with the same user to k, such definition bounds the number of passwords which a network attacker can test to $n * k$ passwords, instead of the optimal number of $\lfloor \frac{n*k}{t+1} \rfloor$ passwords.

Formally, we call a PPSS scheme on dictionary D and secret space S is $(n, t, T, q_U, q_S, \epsilon)$-*secure* if for any $s_0, s_1 \in S$, any set B s.t. $t' \triangleq |\mathsf{B}|$ satisfies $t' \leq t$, and any algorithm $\mathcal{A}$ with running time T, we have

$$|p_0 - p_1| \leq \left\lfloor \frac{q_S}{t - t' + 1} \right\rfloor * \frac{1}{|D|} + \epsilon$$

where p_b is the probability that $\mathcal{A}(s_0, s_1, \mathsf{st}_0, \mathsf{st}_\mathsf{B})$ outputs 1 on access to q_U sessions with oracle $\mathsf{User}^\diamond(p, \mathsf{st}_0)$ and q_S sessions with oracle $\mathsf{Server}^\diamond(\mathsf{st}_{\overline{\mathsf{B}}})$, for st output by $\mathsf{Init}(p, s_b)$ on p chosen at random in D, with probability taken over all random processes.

To make a PPSS scheme easier to use as a building block, e.g. in a construction of a T-PAKE protocol in Section 4, the above security definition needs to be strengthened by allowing the adversary

to learn whether honest user sessions output some reconstructed secret or reject. Intuitively, whenever any application employs a PPSS scheme to reconstruct a secret, it uses this reconstructed secret in some higher-level protocol, e.g. in the case of a T-PAKE to authenticate a key exchange, thus allowing a network adversary to learn whether the PPSS subprotocol accepted or rejected, as in the latter case the protocol built on top of PPSS will visibly diverge from its usual course. Formally, a PPSS scheme on dictionary D and secret space S is $(n, t, T, q_U, q_S, \epsilon)$-*strongly secure* if it satisfies the $(n, t, T, q_U, q_S, \epsilon)$-security definition above with the user oracle $\mathsf{User}^\diamond$ modified so that for every $\mathsf{User}(p, \mathsf{st}_0)$ instance the adversary learns a bit which indicates whether this instance accepts, i.e. outputs some $s' \neq \perp$, or rejects, i.e. outputs $\perp$.

A practical PPSS scheme should satisfy two further properties, namely *robustness* and *soundness*. Robustness requires that a user communicating with n servers recovers the shared secret as long as at least $t + 1$ of these servers follow the protocol. Soundness requires that even n corrupt servers cannot cause a user using the prescribed public parameters st_0 to recover a secret which is different from the one which was initially shared. We differentiate between *weak soundness*, where this is required only for a user entering the correct password, and *strong soundness*, where this is required for any password the user enters, including incorrect ones. Formally a PPSS scheme on dictionary D and secret space S is (T, ϵ)-*robust* if for any $(s, p) \in S \times D$, any B s.t. $n - |\mathsf{B}| \geq t+1$, and any algorithm $\mathcal{A}$ with running time T, the probability that $s' \neq s$ where st is output by $\mathsf{Init}(p, s)$ and s' is output by $\mathsf{User}(p, \mathsf{st}_0)$ interacting with $\mathcal{A}(s, p, \mathsf{st}_\mathsf{B})$ and $\mathsf{Server}^\diamond(\mathsf{st}_{\overline{\mathsf{B}}})$, is bounded by ϵ. To formally define soundness we say that a PPSS scheme on dictionary D and secret space S is (T, ϵ)-*sound* if for any $(s, p, \tilde{p}) \in S \times D \times D$ and any algorithm $\mathcal{A}$ with running time T, the probability that $s' \notin \{s, \perp\}$ where st is output by $\mathsf{Init}(p, s)$ and s' is output by $\mathsf{User}(\tilde{p}, \mathsf{st}_0)$ interacting with $\mathcal{A}(s, p, \tilde{p}, \mathsf{st})$, is bounded by ϵ. We define *weak soundness* in the same way but restricting $\tilde{p}$ to $\tilde{p} = p$.

3. PPSS PROTOCOL SECURE UNDER DDH

We describe the protocol for password protected secret sharing. We start by introducing the basic idea of our scheme, which can use any threshold homomorphic encryption. We then explain how to reduce the round complexity of this basic protocol using special properties of ElGamal. The resulting protocol involves only three message flows between the user and each server (and one round of interaction in an on-line phase), each message involving constant number of group elements. For exposition's sake, we first show this protocol secure only against honest-but-curious players, assuming secure channels (PPSS_1 in Figure 2), and then proceed to explain how we can address active threats and achieve security against malicious adversaries (PPSS_2 in Figure 3).

Basic PPSS Protocol. Our protocol can use any homomorphic encryption with a threshold decryption protocol, but for the sake of subsequent modifications we describe it instantiated with ElGamal. Let g be a generator of group G of prime order q. Take dictionary $D = \mathbb{Z}_q$ and message space $S = G$. Note that any other dictionary can be hashed into $\mathbb{Z}_q$ using a collision-resistant hash, and as we discuss later message space G can be easily extended to a standard message space of fixed-length bitstrings. For $(p, s) \in \mathbb{Z}_q \times G$, procedure $\mathsf{Init}(p, s)$ picks an ElGamal key-pair $(x, y = g^x)$, secret-shares x among servers using a (t, n) secret sharing [23] (i.e. $\mathsf{st}_j = x_j = f(j)$ where f is a random t-degree polynomial over $\mathbb{Z}_q$ s.t. $f(0) = x$), and outputs public parameter st_0 which consists of public key y, a "shifted" ElGamal encryption of password p, $(c_p, d_p) = (g^{r_p}, y^{r_p} g^p)$, and a textbook ElGamal encryption of secret s, $(c_s, d_s) = (g^{r_s}, y^{r_s} s)$, for random r_p, r_s in $\mathbb{Z}_q$. The PPSS protocol between the user U on input $\tilde{p}$ and the servers $\{\mathsf{P}_j\}_{j=1}^n$ on inputs st_j proceeds as follows:

1. U sends an encryption of $\tilde{p}$, $(c_{\tilde{p}}, d_{\tilde{p}}) = (g^{r_{\tilde{p}}}, y^{r_{\tilde{p}}} g^{\tilde{p}})$, to each Server_j. Note that for a legitimate user, $\tilde{p} = p$.
2. P_j's randomize $(c_\delta, d_\delta) = (c_p / c_{\tilde{p}}, , d_p / d_{\tilde{p}})$ by each P_j returning $(c_{\beta_j}, d_{\beta_j}) = ((c_\delta)^{t_j}, (d_\delta)^{t_j})$ for $t_j \xleftarrow{r} \mathbb{Z}_q$.
3. U sends $(c_\beta, d_\beta) = (\prod_{i=1}^n c_{\beta_i}, \prod_{1=1}^n d_{\beta_i})$ to each P_j. Note that (c_β, d_β) is an ElGamal encryption of g^β where $\beta = \sum_{i=1}^n \beta_i = \delta_p \cdot \sum_{i=1}^n t_i$ and $\delta_p = p - \tilde{p}$.
4. P_j's threshold-decrypt $(c_\alpha, d_\alpha) = (c_s \cdot c_\beta, d_s \cdot d_\beta)$ [6], i.e. each P_j returns $z_j = (c_\alpha)^{x_j}$ and U interpolates any $t + 1$ of these z_j's to $z = (c_\alpha)^x$ and outputs $\alpha = d_\alpha \cdot (z)^{-1}$.

Note that $\alpha = s \cdot g^{\sum_i \beta_i} = s \cdot g^{(p-\tilde{p}) \sum_i t_i}$. Hence, $\alpha = s$ if $\tilde{p} = p$; but α is random in G if $\tilde{p} \neq p$ and $\sum_i t_i$ is random in $\mathbb{Z}_q$.

This basic PPSS protocol is secure in the honest-but-curious setting under the DDH assumption on G. Moreover we have to assume secure channels between U and each P_j because $\alpha = s$ can be decrypted from the values (c_β, d_β) and z_j exchanged in the last two messages of this protocol. However, in the presence of malicious parties, and without secure channels, a whole range of issues needs to be addressed: For example, since the protocol relies on the homomorphic property of ElGamal encryption, a malicious user can recover s by setting $(c_{\tilde{p}}, d_{\tilde{p}})$ as a randomization of (c_p, d_p), thus ensuring that $\tilde{p} = p$ without the knowledge of p. In another example, a single malicious server can cancel out all other servers' contributions $(c_{\beta_j}, d_{\beta_j})$ to (c_β, d_β) so that the sum $\sum_i t_i$ hits some adversarially chosen value t. (The malicious server sets its $(c_{\beta_j}, d_{\beta_j})$ as an encryption of g^t divided by the product of all other $(c_{\beta_j}, d_{\beta_j})$'s.) This way the "ciphertext randomization" step (2) above fails its purpose of providing a random mask for s in the case $\tilde{p} \neq p$, and the value $\alpha = s \cdot g^{t(\tilde{p}-p)}$ which the user outputs allows the adversary to recover s in an off-line dictionary attack.

On-Line Non-Interactive PPSS Protocol. Before addressing the active threats, we observe that relying on specific properties of ElGamal Encryption, we can shave off one communication round in the above protocol, by combining the distributed ciphertext randomization step ($i.e.$ step 2) and the threshold decryption step ($i.e.$ step 4), given some precomputation from the Servers. The resulting protocol (Figure 2 below) involves only three communication flows, with the *on-line* phase, i.e. between User contributing its password $\tilde{p}$ and recovering the secret s, taking only two communication flows (hence the term *non-interactive on-line*).

Note that values z and d_β which the user needs to decrypt s in the basic PPSS protocol above are formed as $z = (c_\alpha)^x = (c_s \cdot c_\beta)^x = (g^{r_s + \delta_r \sum_i t_i})^x = y^{r_s} y^{\delta_r \sum_i t_i}$, where $\delta_r \triangleq r_p - r_{\tilde{p}}$, and $d_\beta = \prod_i d_{\beta_i} = y^{\delta_r \sum_i t_i} g^{\delta_p \sum_i t_i}$, where $\delta_p \triangleq p - \tilde{p}$. The decryption works because if $\delta_p = 0$ then $d_\beta = y^{\delta_r \sum_i t_i}$ and $s = d_s \cdot d_\beta \cdot (z)^{-1}$. At first it would seem that the randomization process which creates (c_β, d_β) in steps (2-3) must precede the threshold-decryption step (4), where z is distributively computed as $(c_\alpha)^x = (c_s \cdot c_\beta)^x$. To see how to combine these two steps, first observe that if the protocol is executed not by all n servers but by a subset V of $t + 1$ servers, in particular if $(c_\beta, d_\beta) = (\prod_{i \in V} c_{\beta_i}, \prod_{i \in V} d_{\beta_i})$, and if each P_j in V computes

$$z_j = d_{\beta_j} \cdot (c_s \cdot c_\beta)^{-\lambda_j x_j} \tag{1}$$

where λ_j is a coefficient s.t. $x = \sum_{j \in V} \lambda_j x_j$, then U can still recover s as $s = d_s \cdot \prod_{j \in V} z_j = d_s \cdot d_\beta \cdot (c_s \cdot c_\beta)^{-x}$. Note that P_j can

$\underline{\mathsf{Init}(p,s)}$ (on public parameters g, q, n, t)

$x \xleftarrow{r} \mathbb{Z}_q$, $y \leftarrow g^x$,$\{x_i\}_{i=1}^n \xleftarrow{(n,t)} \mathsf{SS}(x)$, $h \xleftarrow{r} G$, $(r_p, r_s) \xleftarrow{r} (\mathbb{Z}_q)^2$, $(c_p, d_p) \leftarrow (g^{r_p}, y^{r_p} h^p)$, $(c_s, d_s) \leftarrow (g^{r_s}, y^{r_s} s)$
$\mathsf{st}_0 \leftarrow (g, y, h, (c_p, d_p), (c_s, d_s))$, $\{\mathsf{st}_i \leftarrow x_i\}_{i=1}^n$

$\underline{\mathsf{User}(\mathsf{st}_0, \tilde{p}) \rightleftharpoons (\mathsf{Server}_1(\mathsf{st}_0, \mathsf{st}_1), \cdots, \mathsf{Server}_n(\mathsf{st}_0, \mathsf{st}_n))}$

S1 (Server_j) : Pick $t_j \xleftarrow{r} \mathbb{Z}_q$. Compute $(a_j, b_j) \leftarrow (g^{t_j}, (c_p)^{t_j})$. Send (a_j, b_j) to User.

U1 (User) : Pick a set V of $t+1$ servers. Pick $r_{\tilde{p}} \xleftarrow{r} \mathbb{Z}_q$. Compute $\{e_j \leftarrow (a_j)^{r_{\tilde{p}}}\}_{j \in V}$ and $(c_{\tilde{p}}, d_{\tilde{p}}) \leftarrow (g^{r_{\tilde{p}}}, y^{r_{\tilde{p}}} h^{\tilde{p}})$. Compute $c_\beta \leftarrow \prod_{j \in V} (b_j / e_j)$. For all $j \in V$, send $(V, c_\beta, (c_{\tilde{p}}, d_{\tilde{p}}))$ to Server_j.

S2 (Server_j) : Compute $\lambda_j \leftarrow \frac{\prod_{i \in V/\{j\}}(-i)}{\prod_{i \in V/\{j\}}(j-i)} \bmod q$, $w_j \leftarrow (c_s \cdot c_\beta)^{\lambda_j \cdot x_j}$, $d_{\beta,j} \leftarrow (d_p / d_{\tilde{p}})^{t_j}$, and $z_j \leftarrow d_{\beta,j} / w_j$. Send z_j to User.

U2 (User) : Output $s \leftarrow d_s \cdot \prod_{j \in V} z_j$.

Figure 2: PPSS_1: Password-Protected Secret-Sharing secure against Honest-but-Curious Adversaries assuming Secure Channels

compute $d_{\beta_j} = (d_p / d_{\tilde{p}})^{t_j}$ using $d_{\tilde{p}}$ sent produced by U, but since $c_\beta = g^{\delta_r \sum_{j \in V} t_j}$ involves randomization factors $\{t_j\}_{j \in V}$ from all servers in V, computing c_β seems to require the three communication rounds of steps (1-3) above. However, it can be done in two rounds instead if each P_j first sends to U a pair of "randomization commitment" values, $(a_j, b_j) = (g^{t_j}, (c_p)^{t_j}) = (g^{t_j}, g^{r_p t_j})$, which allows U to return to all P_j's the c_β value computed as:

$$c_\beta = \prod_{j \in V} (b_j \cdot (a_j)^{-r_{\tilde{p}}}) \tag{2}$$

This re-organization leads us to the PPSS_1 protocol shown in Figure 2: Each P_j computes its $(g^{t_j}, (c_p)^{t_j})$ randomization commitment in step **S1**. In step **U1** U chooses a set V of $t+1$ servers and sends to each server in V the c_β value computed as in equation 2 together with an encryption of $\tilde{p}$ (technically only sending the $d_{\tilde{p}}$ part of this ciphertext is necessary). In step **S2** each server computes its response z_j as in equation 1, and U in step **U2** decrypts s as $s = d_s \cdot \prod_{j \in V} z_j$. The PPSS_1 protocol in Figure 2 diverges in just one aspect from the above sketch: Namely, the base used in the shifted ElGamal encryption of the password is changed from g to a different random base h. This change is crucial to prevent an off-line dictionary attack that otherwise would be possible since in PPSS_1, unlike the basic PPSS protocol above, the user learns $a = \prod a_i = g^{\sum_i t_i}$. This value, together with user's output $\alpha = s \cdot g^{(p-\tilde{p}) \sum_i t_i}$ computed for an arbitrary $\tilde{p}$, allows an off-line dictionary attack, where each candidate password p' can be tested by checking if $s' = \alpha \cdot (a)^{\tilde{p}-p'}$ looks like a valid plaintext. On the other hand, if the shifted ElGamal encryption encrypts h^p instead of g^p then $\alpha = s \cdot h^{(p-\tilde{p}) \sum_i t_i}$ is a one-time pad encryption of s because under the DDH assumption $h^{\sum_i t_i}$ is a pseudorandom value, and hence so is $h^{\delta_p \sum_i t_i}$ for any $\delta_p \triangleq (p - \tilde{p})$, even given $a_i = g^{\sum_i t_i}$, if $\sum_i t_i$ is random in $\mathbb{Z}_q$.

Security against Malicious Adversaries. Somewhat surprisingly, combining the randomization step with threshold decryption step helps achieve security against malicious adversaries -albeit with additional modifications discussed below. The reason why this collapse of two computation steps makes security argument (and the necessary modifications to the protocol) easier, is that now the contribution of each P_j to threshold-decryption of $\alpha = s \cdot h^{\delta_p \sum_i t_i}$, technically value $w_j = (c_\alpha)^{\lambda_j x_j} = (c_s \cdot c_\beta)^{\lambda_j x_j}$ computed in step **S2**, is output "masked" with P_j's contribution to the "randomization" of the encryption $(c_p / c_{\tilde{p}}, d_p / d_{\tilde{p}})$ of $\delta_p = p - \tilde{p}$, i.e. with the value $d_{\beta,j} = (d_p / d_{\tilde{p}})^{t_j} = y^{\delta_r t_j} h^{\delta_p t_j}$. Intuitively, if $\delta_p \neq 0$, then variable $h^{\delta_p t_j}$ in $d_{\beta,j}$ acts like a pseudorandom mask (assuming DDH) even given values $(a_j, b_j) = (g^{t_j}, g^{r_p t_j})$. This allows us to argue that servers' responses z_j are indistinguishable from random group elements values on sessions where $\delta_p \neq 0$, and thus we are left only with the information released in sessions where $\tilde{p} = p$, i.e. where the adversary guesses the correct password.

It might seem that there is nothing to argue for sessions on which $\tilde{p} = p$, but this is not the case: In a standard password-authenticated protocol (PAKE), the attackers indeed win if they guess the password on any session. However, we have n servers, and an adversary who corrupts $t' \leq t$ of them needs to use the correct password on $t - t' + 1$ sessions with distinct uncorrupted servers in order to break the scheme. Thus the messages output by the first $t - t'$ sessions where adversary uses the correct password must still look indistinguishable from sessions with wrong passwords. Our scheme achieves this property for the following reasons. Firstly we are helped by the properties of Shamir secret-sharing of the decryption key x: If the only sessions on which the servers threshold decryption shares $w_j = (c_\alpha)^{\lambda_j x_j}$ are not masked with pseudorandom values are those where $\tilde{p} = p$ then, by the properties of Shamir secret sharing, the first $t - t'$ such shares produced by the uncorrupted servers, even given the t' shares x_j of corrupted servers, are still statistically independent of the decryption key x. Consequently, as long as $\tilde{p} = p$ on sessions with no more than $t - t'$ uncorrupted servers, adversary's interaction with the scheme effectively never uses the real decryption key x. Therefore in particular, until that point we can look at tuple (g, y, g^{t_j}, y^{t_j}) for each uncorrupted P_j as a DDH tuple, in which case even if $\delta_p = 0$, the value $d_{\beta,j} = y^{\delta_r t_j} h^{\delta_p t_j} = y^{\delta_r t_j} = y^{(r_p - r_{\tilde{p}}) t_j}$ still acts like a pseudorandom one-time pad masking the partial decryption w_j, even given $(a_j, b_j) = (g^{t_j}, g^{r_p t_j})$, provided that $r_{\tilde{p}} \neq r_p$ (which is easy to prevent, see below). Here we list the remaining differences between the resulting protocol, PPSS_2 in Figure 3, and PPSS_1 in Figure 2:

(1) A syntactic change is that in Figure 3 we use Server_j to denote the j-th *session* of the Server algorithm, i.e. $j \in \{1, ..., q_S\}$, and we use ID_j to denote the ID of the Server who executes this session, i.e. $ID_j \in \{1, ..., n\}$. This notation makes it clearer how the protocol executes in the concurrent setting, where each server runs many sessions of the Server algorithm.

(2) Another minor change is that the user pre-computes the λ_j coefficient for each server session in V. This is to keep the size of user's message constant. (The user could also compute $Q_j = (c_s \cdot c_\beta)^{\lambda_j}$ in step **S1** and send it instead of (λ_j, c_β), then Server_j would compute $w_j \leftarrow (Q_j)^{x_{ID_j}}$ and both parties would use Q_j in the proof π_{3j}. This change would have no effect on security and it would shift one exponentiation from the server to the user.)

(3) In PPSS_1 the server's responses z_i are sent back in cleartext,

$\underline{\mathsf{Init}(p,s)}$ (on public parameters g, q, n, t)

$x \xleftarrow{r} \mathbb{Z}_q$, $y \leftarrow g^x$, $\{x_i\}_{i=1}^n \xleftarrow{(n,t)} \mathsf{SS}(x)$, $(h, \hat{g}, \hat{h}, \hat{y}, \bar{g}) \xleftarrow{r} (G)^4$, $(r_p, r_s) \xleftarrow{r} (\mathbb{Z}_q)^2$,$(c_p, d_p) \leftarrow (g^{r_p}, y^{r_p} h^p)$, $(c_s, d_s) \leftarrow (g^{r_s}, y^{r_s} s)$
$\{r_i \leftarrow \mathbb{Z}_q; y_i \leftarrow g^{x_i} h^{r_i}\}_{i=1}^n$, $\mathsf{st}_0 \leftarrow (g, h, y, \{y_i\}_{i=1}^n, \hat{g}, \hat{h}, \hat{y}, \bar{g}, (c_p, d_p), (c_s, d_s))$, $\{\mathsf{st}_i \leftarrow (x_i, r_i)\}_{i=1}^n$

$\underline{\mathsf{User}(\mathsf{st}_0, \tilde{p}) \rightleftharpoons \{\mathsf{Server}_j(\mathsf{st}_0, \mathsf{st}_{ID_j})\}_{j=1}^n}$ (ID_j denotes Server ID of j-th Server session, $ID_j \in \{1, ..., n\}$.)

S1 (Server_j) : Pick $t_j \xleftarrow{r} \mathbb{Z}_q$. Compute $(a_j, b_j, \bar{a}_j) \leftarrow (g^{t_j}, (c_p)^{t_j}, (\bar{g})^{t_j})$, and $\pi_{1j} \leftarrow \mathcal{P}[\mathcal{L}_{S1}^{\mathsf{st}_0}]((a_j, b_j, \bar{a}_j), t_j)$.
Send $(ID_j, a_j, b_j, \bar{a}_j, \pi_{1j})$ to User.

U1 (User) : Pick set V of $t+1$ sessions $\{\mathsf{Server}_j\}_{j \in V}$ run by distinct servers s.t. $\{\mathcal{V}[\mathcal{L}_S^{\mathsf{st}_0}]((a_j, b_j, \bar{a}_j), \pi_{1j}) = 1\}_{j \in V}$.
Pick $r_{\tilde{p}} \xleftarrow{r} \mathbb{Z}_q$.

Compute: $\{e_j \leftarrow (a_j)^{r_{\tilde{p}}}\}_{j \in V}$, $c_\beta \leftarrow \prod_{j \in V} (b_j / e_j)$, $(c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}) \leftarrow (g^{r_{\tilde{p}}}, y^{r_{\tilde{p}}} h^{\tilde{p}}, (\hat{g})^{r_{\tilde{p}}}, (\hat{y})^{r_{\tilde{p}}} (\hat{h})^{\tilde{p}})$
$\{\lambda_j \leftarrow \frac{\prod_{i \in V/\{j\}} (-ID_i)}{\prod_{i \in V/\{j\}} (ID_j - ID_i)} \bmod q$, $\pi_{2j} \leftarrow \mathcal{P}[\mathcal{L}_U^{\mathsf{st}_0}]((a_j, e_j, c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}), (r_{\tilde{p}}, \tilde{p}))\}_{j \in V}$

Send $(\lambda_j, c_\beta, e_j, (c_{\tilde{p}}, d_{\tilde{p}}), (\hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}), \pi_{2j})$ to Server_j.

S2 (Server_j) : Stop if $\mathcal{V}[\mathcal{L}_U^{\mathsf{st}_0}]((a_j, e_j, c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}), \pi_{2j}) = 0$. Pick $r_{z_j} \leftarrow \mathbb{Z}_q$.

Compute: $w_j \leftarrow (c_s \cdot c_\beta)^{\lambda_j \cdot x_{ID_j}}$, $d_{\beta,j} \leftarrow (d_p / d_{\tilde{p}})^{t_j}$, $z_j \leftarrow d_{\beta,j} / w_j$, $(c_{z_j}, d_{z_j}) \leftarrow (g^{r_{z_j}}, (c_{\tilde{p}})^{r_{z_j}} \cdot z_j)$
$\pi_{3j} \leftarrow \mathcal{P}[\mathcal{L}_{S2}^{\mathsf{st}_0, ID_j}]((c_{z_j}, d_{z_j}, c_{\tilde{p}}, a_j, d_p / d_{\tilde{p}}, (c_s \cdot c_\beta)^{\lambda_j}), (r_{z_j}, t_j, x_{ID_j}, r_{ID_j}))$

Send $((c_{z_j}, d_{z_j}), \pi_{3j})$ to User.

U2 (User) : Stop if $\exists_{j \in V}$, s.t. $\mathcal{V}[\mathcal{L}_{S2}^{\mathsf{st}_0, ID_j}]((c_{z_j}, d_{z_j}, c_{\tilde{p}}, a_j, d_p / d_{\tilde{p}}, (c_s \cdot c_\beta)^{\lambda_j}), \pi_{3j}) = 0$.
Output $s \leftarrow d_s \cdot (\prod_{j \in V} d_{z_j}) / (\prod_{j \in V} c_{z_j})^{r_{\tilde{p}}}$.

Figure 3: PPSS_2: Password-Protected Secret-Sharing Protocol Secure against Malicious Adversaries

hence the protocol is insecure even with a passive eavesdropper on the channels between the user and the servers. To counter such eavesdropping attack, each server encrypts its response z_j in step **S2** using ElGamal encryption with user's value $c_{\tilde{p}} = g^{r_{\tilde{p}}}$ sent in step **U1** serving as an ElGamal public key, and the user will use $r_{\tilde{p}}$ to decrypt these ciphertexts in step **U2**. Even though $c_{\tilde{p}}$ is used as first part of user's ElGamal ciphertext $(c_{\tilde{p}}, d_{\tilde{p}})$ encrypting $\tilde{p}$, it turns out that we can re-use it for encrypting information from servers to the user. As for preventing more sophisticated man-in-the-middle attacks, both the user's and the servers' messages are accompanied by simulation-sound zero-knowledge proofs of well-formedness, and one consequence of these proofs is that they make it difficult for the man-in-the-middle adversary to re-use the messages from the honest parties.

(4) To enable efficient simulation of the honest parties in the protocol (and thus show that an attacker learns no useful information in an interaction with these parties), we could amend each protocol message with a proof of knowledge of the randomness used to create it (e.g. t_j in server's message **S1** or $r_{\tilde{p}}$ in user's message **U1**), but this would require concurrently extractable zero-knowledge proofs of knowledge, and even the most efficient such proofs, e.g. [10], would make the protocol significantly more expensive. However, the simulator turns out to need to compute only values which include adversarial players' randomness in the exponent, i.e. values of the form z^w where w is the randomness of the adversarially controlled party, and z is some group element known to the simulator. Therefore we can avoid the need for proofs of knowledge with a technique used before e.g. by Gennaro and Shoup [25]: If the simulator needs a value of the form z^w in computing a (simulated) response to some message, we extend that message by a value of the form $(g')^w$ where g' is an additional random group element in the parameter string, and include a zero-knowledge (and simulation-sound) proof that the message is formed using consistent randomness w. The simulator will then compute z^w by embedding z into g'. Such proofs, as opposed to concurrent proofs of knowledge, are inexpensive, and they can be made non-interactive in the Random Oracle Model. This is why we extend server's message **S1** with value $\bar{a}_j = (\bar{g})^{t_j}$, and user's message **U1** with value $\hat{c}_{\tilde{p}} = (\hat{g})^{r_{\tilde{p}}}$. (This is also why the secondary encryption of $\tilde{p}$ we discuss in the next item uses a new base $\hat{h}$ instead of h.)

(5) It seems difficult to simulate this protocol efficiently unless the simulator can test whether the user's ciphertext encrypts the correct password, e.g. because the simulator can replace the servers' responses by random values, as argued above, only if the user encrypts an incorrect password. However, since the adversary's view includes an encryption of the correct password, the DDH reduction which shows that this encryption is semantically secure cannot know the corresponding decryption key, and thus cannot perform such test. We overcome this quandary using the twin encryption paradigm used in the design of CCA-secure encryption or CMA-secure signatures starting with [21, 13]. Namely, we extend the user's message by a "secondary" encryption of $\tilde{p}$, under an independent ElGamal public key $\hat{y}$. For technical reasons it turns out that this encryption can re-use the same randomness $r_{\tilde{p}}$ used in the "primary" ciphertext $(c_{\tilde{p}}, d_{\tilde{p}})$, and so it suffices to extend the **U1** message by just one value $\hat{d}_{\tilde{p}} = (\hat{y})^{r_{\tilde{p}}} (\hat{h})^{\tilde{p}}$. The simulator can now test whether $(c_{\tilde{p}}, d_{\tilde{p}})$ encrypts $\tilde{p} = p$ by decrypting $(\hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}})$ – since by the soundness of the user's proof these two ciphertexts encrypt the same plaintext – using a trapdoor key $\hat{x} = DL(\hat{g}, \hat{y})$ which is independent of $x = DL(g, y)$. In this way the simulator can test for password correctness even as it embeds a DDH challenge into the primary key y and the public ciphertexts (c_p, d_p) and (c_s, d_s).

(6) For verifiability of Server_j's computation of the (encrypted) response z_j in step **S2**, we amend the public parameters with the set of Pedersen commitments $\{y_i = g^{x_i} h^{r_i}\}_{i=1}^n$ to the server's shares $\{x_i\}_{i=1}^n$ of the decryption key. Note that the Server_j sessions where the adversary guesses the correct password effectively implement an exponentiation oracle $z_z = F_j(m) = m^{x_{ID_j}}$, e.g. setting $c_\beta = m^{1/\lambda_j} / c_s$. Therefore the adversary who happens

to guess the password can adaptively choose t servers to query in this way (assume for simplicity that no servers are corrupted). This presents a problem which arises in adaptively secure threshold cryptosystems, e.g. [5]. Namely, if the simulator had to commit itself to n uncorrupted shares using a simpler perfectly binding commitment scheme, $y_i = g^{x_i}$, then it could consistently respond to such queries only by guessing at the beginning of the interaction the set of t servers which the adversary later chooses to query in this way. (Note that the simulator has to answer such queries correctly, as the adversary could test if $F_j(g^a) = (y_{ID_j})^a$.) This is why we use Pedersen commitments instead.

Non-Interactive Simulation-Sound Zero-Knowledge Proofs. To assure that protocol messages are well-formed we use simulation-sound non-interactive zero-knowledge (SS-NIZK) proofs [22], a slightly weaker notion than non-malleable NIZK's. Our protocols could also use interactive version of such proofs, which can be realized with comparable efficiency without Random Oracles – see e.g. [12], but using non-interactive proofs enables best round complexity and keeps a protocol write-up simple. Since these are standard notions, we recall only briefly that a NIZK proof system for language $\mathcal{L}$ is a triple of algorithms, prover $\mathcal{P}$ which produces a proof π on input a statement instance x and witness w, verifier $\mathcal{V}$ which accepts or rejects on input (x, π), and a simulator $\mathcal{S}$ which outputs a (simulated) proof on just the input x, using some trapdoors embedded in public parameters (or intercepting honest players' interaction with a hash function in ROM). We will use an exact security version of the SS-NIZK notion, calling such proof system $(T_S, q_P, \epsilon_{ZK}, \epsilon_{SS})$ *simulation-sound zero-knowledge* (SSZK) if there is a simulator algorithm $\mathcal{S}$ running in time T_S which answers up to q_P prover queries with simulated proofs on adaptively chosen statements of adversary's choice (which can include false statements) s.t. (1) the statistical difference between the view of an interaction with $\mathcal{S}$ and an interaction with the real prover is at most ϵ_{ZK}, and (2) the probability that any adversary interacting with $\mathcal{S}$ outputs a correct proof on a new false statement, i.e. a different from those for which it receives a simulated proof from $\mathcal{S}$, is at most ϵ_{SS}. We need such proofs for three languages corresponding to the three protocol messages, all parameterized by public parameters $\mathsf{st}_0 = (g, h, y, \{y_i\}_{i=1}^n, \hat{g}, \hat{h}, \hat{y}, \bar{g}, (c_p, d_p), (c_s, d_s))$:

$$\mathcal{L}_{S1}^{\mathsf{st}_0} = \{(a, b, \bar{a}) \in (G)^3 \mid \exists\, t \in \mathbb{Z}_q \;\; s.t. \\ (a, b, \bar{a}) = (g^t, (c_p)^t, (\bar{g})^t)\}$$

$$\mathcal{L}_{U}^{\mathsf{st}_0} = \{(a, e, c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_p) \in (G)^6 \mid \exists\, (r_{\tilde{p}}, \tilde{p}) \in (\mathbb{Z}_q)^2 \;\; s.t. \\ (e, c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}) = (a^{r_{\tilde{p}}}, g^{r_{\tilde{p}}}, y^{r_{\tilde{p}}} h^{\tilde{p}}, (\hat{g})^{r_{\tilde{p}}}, (\hat{y})^{r_{\tilde{p}}} (\hat{h})^{\tilde{p}})\}$$

$$\mathcal{L}_{S2}^{\mathsf{st}_0,i} = \{(c_z, d_z, c_{\tilde{p}}, a, \Delta, P) \in (G)^6 \mid \exists\, (r_z, t, x, r) \in (\mathbb{Z}_q)^3 \;\; s.t. \\ (y_i, a, c_z, d_z) = (g^x h^r, g^t, g^{r_z}, (c_{\tilde{p}})^{r_z} \Delta^t P^{-x})\}$$

These languages involve equality relations on representations of some group elements in bases defined by other group elements. Well-known simulation-sound NIZKs for such relations in ROM are generalizations of Schnorr's proof of discrete logarithm knowledge and Chaum's proof of discrete logarithm equality, using the Fiat-Shamir heuristic, see e.g. [4]. The prover and the verifier in these proofs perform as many (multi)-exponentiations as are used to define the corresponding language, e.g. three for $\mathcal{L}_{S1}^{\mathsf{st}_0}$, five for $\mathcal{L}_{U}^{\mathsf{st}_0}$, and four for $\mathcal{L}_{S2}^{\mathsf{st}_0,i}$. (The verifier's computation can be reduced further using the techniques of batch verification of discrete-log based signatures, e.g. [2].) These proof systems achieve simulation sound zero knowledge with error bounds $\epsilon_{ZK} = (q_P \cdot q_H)/q$ and $\epsilon_{SS} = q_H/q$, where q_H is the upper bound on adversary's hash function queries, with the simulators whose running time is the same as that of the corresponding provers.

Efficiency, Standard Message Spaces, Soundness, Robustness. *Efficiency:* For efficiency estimates we assume that the SS-NIZK proofs are implemented in ROM, with proof verification implemented using multi-exponentiation and randomization of each verification equation, as in signature batch-verification of [2]. (This increases the soundness errors of these proofs by negligible amounts.) We count multi-exponentiations involving up to 5 bases as single "multiexp" operations, e.g. we estimate the verification costs of proofs $\pi_{1j}, \pi_{2j}, \pi_{3j}$ as 2, 4, and 3 multiexp's, respectively. To reduce user's costs we also arrange the $t+1$ proofs π_{2j} into a single proof showing consistency of $(c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}})$ ciphertexts and $t+1$ proofs of consistency of each (a_j, e_j) pair with $c_{\tilde{p}}$. Under these assumptions the protocol costs $8t+17$ multiexp's for the user and 16 for each server. Using precomputation we can reduce the on-line cost to $7t+8$ multiexp's for the user and 6 for each server.

Message Space: Protocol PPSS_2 works on a non-standard message space, i.e. s must be an element of group G, but this can be easily changed to standard message spaces: Let l be a keylength of semantically secure symmetric encryption, and let G be a DDH group of order q where $|q| = 3l$. The public parameters string st_0 is amended by a random key k of a universal hash function H_k mapping elements of G onto l-bit bitstrings s.t. $H_k(x)$ is statistically indistinguishable from random 160-bit string if x is random in G. A modified PPSS protocol on password p and secret m, which is now any bitstring, runs the PPSS_2 protocol of Figure 3 on a random element s in G, and attaches to the public parameters st_0 the hash key k and a symmetric encryption of m under key $H_k(s)$. In the recovery protocol the user hashes the recovered value s to decrypt the shared secret m. Note that we can encrypt directly under $H(s)$ with a subgroup of size $|q| = \max(l, 160)$ assuming either a "Hashed Diffie-Hellman" assumption [1] or just DDH but modeling H as a random oracle.

Soundness: The PPSS protocol of Figure 3 satisfies only weak soundness, i.e. even malicious servers cannot make the user who runs on the correct password recover any other secret except of that which was initially shared. This follows from soundness of the SS-NIZK proof systems used in servers' messages, since these proofs effectively force the servers to perform the prescribed protocol. One easy way to extend this to strong soundness is to include a public key pk of an existentially unforgeable signature scheme in st_0, use the corresponding signature key to sign secret s, and run the PPSS algorithm as modified above on a concatenation of s with this signature. The protocol then proceeds as above except that at the end the user parses the recovered secret as $s|\sigma$ and outputs s only if σ is a valid signature on s under the key pk in st_0.

Robustness: Note that the PPSS protocol of Figure 3 does not satisfy robustness, because the user outputs $\perp$ if some server sessions in set V fail to send back correct responses in step **S2**. However, robustness can be achieved by modifying the PPSS_2 protocol as follows: If some sessions in V fail in step **S2**, the user marks the servers executing them as "dishonest" and restarts the protocol using a fresh set of $t+1$ server sessions, executed by servers which have not been marked. In the presence of $r \leq n-(t+1)$ active faults, this can take up to r rounds of interaction before all server sessions the user chooses are with non-faulty servers, which guarantees recovery of s. In practice such active attacks should be extremely rare because the attack reveals the servers corrupted by the adversary, in which case the user should eliminate, or clean-up, the attacked server(s) and re-initialize the protocol. Hence the adversary would achieve only a momentary slowdown of the secret-

reconstruction protocol at a price of expulsion from the servers it has managed to corrupt. Random network errors are of greater concern, but in practice the user should ping each server before including its session in set V in step **U1**. Note that the user could compute its **U1** messages for n instead of $t+1$ distinct servers and only then settle on the set V, and the server's response **S2** involves only three multi-exponentiations computed on-line, namely d_{z_j}, verification of π_{2j}, and the fourth element of π_{3j}, which involves a flexible base. Thus at the price of increasing **U1**'s cost by a factor of $\frac{n}{t+1}$ we can reduce the computational cost contributing to the delay between the user's ping of the servers in step **U1** and receiving their responses, to three on-line multiexp's needed in step **S2**.

THEOREM 1. *Let G be a group of prime order q where the DDH problem is $(T_{ddh}, \epsilon_{ddh})$-hard, let $t_{\exp}$ be the time of full (multi)-exponentiation in G, let $\Pi[\mathcal{L}_U^{\mathsf{st}_0}]$, $\Pi[\mathcal{L}_{S1}^{\mathsf{st}_0}]$, and $\Pi[\mathcal{L}_{S2}^{\mathsf{st}_0,i}]$ be $(T_S, q_P, \epsilon_{ZK}, \epsilon_{SS})$-SSZK proof systems. Protocol PPSS_2 is a $(n, t, T, q_U, q_S, \epsilon)$* strongly secure *PPSS scheme on message space G and dictionary $D \subseteq \mathbb{Z}_q$, as long as $\max(q_U, n \cdot q_S) \leq q_P$ and*

$$\begin{aligned} T &\leq T_{ddh} - 3T_S - (10 + n + (8t+12)q_U + 10nq_S) \cdot t_e \\ \epsilon &\leq 6\epsilon_{ZK} + (5q_U + 3nq_S + 4)(\epsilon_{ddh} + \epsilon_{SS}) \end{aligned}$$

PROOF. Let $\mathcal{A}$ be an algorithm followed by an adversary attacking the PPSS_2 scheme, running in time T, accessing at most q_U user and q_S server sessions, and corrupting servers $\{\mathsf{P}_i\}_{i\in\mathsf{B}}$ for some set B s.t. $|\mathsf{B}| = t' \leq t$. Figure 4 describes a series of games, $\mathsf{G}_0, ..., \mathsf{G}_8$, all initialized on secret s, where G_0 models the interaction of $\mathcal{A}$ with the PPSS_2 scheme, with slight modifications explained below, while games $\mathsf{G}_1, ..., \mathsf{G}_8$ are modifications of G_0 used in the security argument below. We first explain how the game G_0 models the interaction of $\mathcal{A}$ with the PPSS_2 scheme. The $\mathsf{Init}^\diamond$ procedure treats threshold (t, n) and the set of corrupted players B as parameters. Note that adversary $\mathcal{A}$ can engage in at most q_S sessions but it is up to $\mathcal{A}$ to decide which servers will be involved in these sessions. We handle this in the security game by creating q_S distinct sessions for each server, thus $n \cdot q_S$ total sessions, even though a q_S-limited adversary will utilize only q_S of them. We denote the identity of the server executing the j-th session, where $j = 1, .., n \cdot q_S$, as ID_j, which can be set w.l.o.g. as $ID_j = (j \bmod n) + 1$. Procedure $\mathsf{Init}^\diamond$ on input s picks $p \xleftarrow{r} D$ and follows the real initialization procedure $\mathsf{Init}(p, s)$ in generating the vector of initial states st, including the parameters st_0 and the states st_i for each server P_i. $\mathsf{Init}^\diamond$ also executes the first step **S1** for all server sessions, and hands off to $\mathcal{A}$ the parameters st_0, the shares of corrupted servers $\mathsf{st}_B = \{x_i\}_{i\in B}$, and the first message on all nq_S sessions $\{\mathsf{Server}_j\}_{j=1}^{nq_S}$. Then $\mathcal{A}$ can make q_U queries to the $\mathsf{User}^\diamond$ oracle, which models execution of step **U1** of procedure $\mathsf{User}(\mathsf{st}_0, p)$, and q_S queries to the execution of step **S2** of the j-th sesssion of the Server procedure. Note that the inputs $\mathcal{A}$ provides to these oracles must syntactically conform to, respectively, a set of **S1** messages from some $t+1$ servers for the $\mathsf{User}^\diamond$ oracle, and user **U1** message in the case of the $\mathsf{Server}_j^\diamond$ oracle, but the adversary can choose these values at wish, in particular they do not have to equal to the corresponding variables produced by the oracles simulating the honest players.

Proof Roadmap. We will use the following notation: By $\mathsf{F}_{i,s}$ we denote event F defined in Figure 4 happening in the interaction between $\mathcal{A}$ and game G_i initialized on secret s, writing F_i if s is not even present in that game. By $p_i(s)$ we denote $\Pr[1 \leftarrow (\mathcal{A} \rightleftharpoons \mathsf{G}_i(s))]$. Since G_0 differs from the real interaction, we denote the corresponding probability in the real interaction as $\bar{p}_0(s)$. The intuition for the security proof is that the event F roughly corresponds to an adversary sending an encryption of the correct password on at least $t - t' + 1$ sessions run by distinct servers, in which case $\mathcal{A}$ could indeed learn s. The goal of the security argument is to show that for q_S, q_U, T satisfying the bounds in the theorem claim, and any s, s', it holds that

$$|\bar{p}_0(s) - \bar{p}_0(s')| \leq \lfloor \frac{q_S}{t - t' + 1} \rfloor * \frac{1}{|D|} + \epsilon' \qquad (3)$$

where $\epsilon' = 6\epsilon_{ZK} + (5q_U + 3nq_S + 4)(\epsilon_{ddh} + \epsilon_{SS})$ This would follow if we showed that $|\bar{p}_0(s_1) - \bar{p}_0(s_0)| \leq \Pr[\mathsf{F}_8] + \epsilon'$ and $\Pr[\mathsf{F}_8] \leq \lfloor \frac{q_S}{t-t'+1} \rfloor * \frac{1}{|D|}$, and below we show this by showing indistinguishability of adversarial views each consecutive games, with the last game, G_8, releasing no information about either p or s, implying the above probability bound on F_8.

Let $p_i^{\neg\mathsf{F}}(s)$ denotes the joint probability $\Pr[1 \leftarrow (\mathcal{A} \rightleftharpoons \mathsf{G}_i(s)) \wedge \neg\mathsf{F}_{i,s}]$; let t_e be a time of a single (multi)-exponentiation in G; let the DDH problem be $(T_{ddh}, \epsilon_{ddh})$-hard in group G; and let proof systems $\Pi[\mathcal{L}_U^{\mathsf{st}_0}]$, $\Pi[\mathcal{L}_{S1}^{\mathsf{st}_0}]$ and $\Pi[\mathcal{L}_{S1}^{\mathsf{st}_0}]$ be $(T_S, q_P, \epsilon_{ZK}, \epsilon_{SS})$ simulation-sound zero-knowledge. Denote $T_{red} = (10+n+(8t+12)q_U + 10nq_S) \cdot t_e$: This is the maximum computational cost, in addition to simulating the ZK proofs, encountered by any of our reductions below. Assume that the bounds on q_U, q_S, T are satisfied as in the theorem claim, namely $\max(q_U, n \cdot q_S) \leq q_P$ and $T + (T_{red} + 3T_S) \leq T_{ddh}$. We will show the following four facts, for any s, s', for $\epsilon_{0,3} = 3\epsilon_{ZK} + (2q_U + nq_S)(\epsilon_{ddh} + \epsilon_{SS})$, $\epsilon_{4,5} = \epsilon_{ddh}$, and $\epsilon_{4,8} = (q_U + nq_S + 2)(\epsilon_{ddh} + \epsilon_{SS})$.

CLAIM 1. $|\bar{p}_0(s) - p_3(s)| \leq \epsilon_{0,3}$

CLAIM 2. $|p_3(s) - p_3(s')| \leq |p_4^{\neg\mathsf{F}}(s) - p_4^{\neg\mathsf{F}}(s')| + \max(\Pr[\mathsf{F}_{4,s}], \Pr[\mathsf{F}_{4,s'}])$

CLAIM 3. $\Pr[\mathsf{F}_{4,s}] \leq \lfloor \frac{q_S}{t-t'+1} \rfloor * \frac{1}{|D|} + \epsilon_{4,8}$

CLAIM 4. $|p_4^{\neg\mathsf{F}}(s) - p_4^{\neg\mathsf{F}}(s')| \leq 2\epsilon_{4,5}$

Summing these up we obtain that $|\bar{p}_0(s) - \bar{p}_0(s')| \leq 2(\epsilon_{0,3} + \epsilon_{4,5}) + \epsilon_{4,8} + \lfloor \frac{q_S}{t-t'+1} \rfloor * \frac{1}{|D|}$, which implies inequality 3, because $2(\epsilon_{0,3} + \epsilon_{4,5}) + \epsilon_{4,8} \leq \epsilon'$. Now, claim 1 follows from claims 5-7 below, because $\epsilon_{0,3}$ is the sum of the upper bounds on the distances $|p_{i-1}(s) - p_i(s)|$ for $i = 1, 2, 3$ in claims 5-7 below, and $|\bar{p}_0(s) - p_0(s)|$, which is bounded by $3\epsilon_{ZK}$. Claim 2 follows from claim 8 below: Note that $|p_3(s) - p_3(s')| = |(p_3^{\neg\mathsf{F}}(s) + p_3^{\mathsf{F}}(s)) - (p_3^{\neg\mathsf{F}}(s') + p_3^{\mathsf{F}}(s'))|$ is upper bounded by $|p_3^{\neg\mathsf{F}}(s) - p_3^{\neg\mathsf{F}}(s')| + \max(\Pr[\mathsf{F}_{3,s}], \Pr[\mathsf{F}_{3,s'}])$; and moreover by claim 8, it follows that $p_3^{\neg\mathsf{F}}(s) = p_4^{\neg\mathsf{F}}(s)$ and $\Pr[\mathsf{F}_{3,s}] = \Pr[\mathsf{F}_{4,s}]$ for all s. Claim 3 follows from claims 9-12 below, because (1) $\epsilon_{4,8}$ is the sum of the upper bounds on the distances $|\Pr[\mathsf{F}_{i-1,s}] - \Pr[\mathsf{F}_{i,s}]|$ shown for $i = 5, 6, 7, 8$ in claims 9-12, and (2) $\Pr[\mathsf{F}_{8,s}] \leq \lfloor \frac{q_S}{t-t'+1} \rfloor * \frac{1}{|D|}$ because G_8 releases no information about p. Claim 4 follows from claim 9 below because game G_5 is independent of secret s, and therefore for every s, s' we have that $p_5^{\neg\mathsf{F}}(s) = p_5^{\neg\mathsf{F}}(s')$.

CLAIM 5. *Games G_0 and G_1 are indistinguishable under DDH. Concretely, $|p_0(s) - p_1(s)| \leq q_U(\epsilon_{ddh} + \epsilon_{SS})$.*

PROOF. To show that G_0 and G_1 are indistinguishable, we make a hybrid argument over q_U user sessions. We define a series of intermediary games G_0^i, between G_0 and G_1, where G_0^i follows G_1 in the $\mathsf{User}^\diamond$ oracle calls on the first i user sessions, *i.e.* it picks $\hat{d}_{\tilde{p}} \xleftarrow{r} G$, and then follows G_0 on the remaining sessions. Clearly, $\mathsf{G}_0^0 \equiv \mathsf{G}_0$ and $\mathsf{G}_0^{q_U} \equiv \mathsf{G}_1$. Let $p_0^i = \Pr[1 \leftarrow (\mathcal{A} \rightleftharpoons \mathsf{G}_0^i)]$. For each $i > 0$ we construct reduction $\mathcal{R}_{0,1}^i$ which on input a DDH challenge $(A, B, C) = (g^a, g^b, g^c)$ follows G_0 during $\mathsf{Init}^\diamond$ except

$\mathsf{Init}^{\diamond}$(on input s)

$x \xleftarrow{r} \mathbb{Z}_q, y \leftarrow g^x, \{x_i\}_{i=1}^n \overset{(n,t)}{\longleftarrow} \mathsf{SS}(x)$	$y \xleftarrow{r} G, \{x_i\}_{i=1}^n \overset{(n,t)}{\longleftarrow} \mathsf{SS}(0)$	G_4
$(h, \hat{g}, \hat{h}, \bar{g}) \xleftarrow{r} (G)^3$		
$\hat{y} \xleftarrow{r} G$	$\hat{x} \xleftarrow{r} \mathbb{Z}_q, \hat{y} \leftarrow (\hat{g})^{\hat{x}}$	G_3
$r_s \xleftarrow{r} \mathbb{Z}_q, (c_s, d_s) \leftarrow (g^{r_s}, y^{r_s} s)$	$(c_s, d_s) \xleftarrow{r} (G)^2$	G_5
$p \leftarrow D, r_p \xleftarrow{r} \mathbb{Z}_q, (c_p, d_p) \leftarrow (g^{r_p}, y^{r_p} h^p)$	$(c_p, d_p) \xleftarrow{r} (G)^2$	G_8
$\mathsf{idSet} \leftarrow \{\}, \mathsf{CpSet} \leftarrow \{\}, \mathsf{F} \leftarrow \mathsf{false}$ $\{\ t_j \xleftarrow{r} \mathbb{Z}_q, (a_j, b_j, \bar{a}_j) \leftarrow (g^{t_j}, (c_p)^{t_j}, \bar{g}^{t_j}), \pi_{1j} \leftarrow \mathcal{S}[\mathcal{L}_{S1}^{\mathsf{st}_0}](a_j, b_j, \bar{a}_j)\ \}_{j=1}^{n \cdot q_S}$ $\{\ r_i \xleftarrow{r} \mathbb{Z}_q,\ \mathsf{st}_i \leftarrow (x_i, r_i),\ y_i \leftarrow g^{x_i} h^{r_i}\ \}_{i=1}^n$ $\mathsf{st}_0 \leftarrow (g, h, y, \{y_i\}_{i=1}^n, \hat{g}, \hat{h}, \hat{y}, \bar{g}, (c_p, d_p), (c_s, d_s)),\ \mathsf{Ret}(\mathsf{st}_0, \{\mathsf{st}_i\}_{i \in \mathsf{B}}, \{a_j, b_j, \bar{a}_j, \pi_{1j}\}_{j=1}^{n \cdot q_S})$		

$\mathsf{User}^{\diamond}(\{ID_j, a_j, b_j, \bar{a}_j, \pi_{1j}\}_{j=1}^{t+1})$

If $(\exists_{j \in \{1,..,t+1\}}\ \mathsf{st}\ \mathcal{V}[\mathcal{L}_{S1}^{\mathsf{st}_0}]((a_j, b_j, \bar{a}_j), \pi_{1j}) = 0)$ then ABORT this session. $r_{\tilde{p}} \xleftarrow{r} \mathbb{Z}_q, (c_{\tilde{p}}, \hat{c}_{\tilde{p}}) \leftarrow (g^{r_{\tilde{p}}}, (\hat{g})^{r_{\tilde{p}}}), \mathsf{CpSet} \leftarrow \mathsf{CpSet} \cup \{c_{\tilde{p}}\}, \{e_j \leftarrow (a_j)^{r_{\tilde{p}}}\}_{j=1}^{t+1}$		
$d_{\tilde{p}} \leftarrow y^{r_{\tilde{p}}} h^p$	$d_{\tilde{p}} \xleftarrow{r} G$	G_7
$\hat{d}_{\tilde{p}} \leftarrow \hat{y}^{r_{\tilde{p}}} (\hat{h})^p$	$\hat{d}_{\tilde{p}} \xleftarrow{r} G$	G_1
$\{\pi_{2j} \leftarrow \mathcal{S}[\mathcal{L}_U^{\mathsf{st}_0}](a_j, e_j, c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}})\}_{j=1}^{t+1},\ \mathsf{Ret}(\{e_j\}_{j=1}^{t+1}, (c_{\tilde{p}}, d_{\tilde{p}}), (\hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}), \{\pi_{2j}\}_{j=1}^{t+1})$		

$\mathsf{Server}_j^{\diamond}(\lambda_j, c_\beta, e_j, c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}, \pi_{2j})$

If $(\mathcal{V}[\mathcal{L}_U^{\mathsf{st}_0}]((a_j, e_j, c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}), \pi_{2j}) = 0)$ then ABORT this session. If $(\hat{d}_{\tilde{p}}/(\hat{c}_{\tilde{p}})^{\hat{x}} = (\hat{h})^p)$ then $\mathsf{idSet} \leftarrow \mathsf{idSet} \cup \{ID_j\}$; If $\lvert\mathsf{idSet}\rvert > t - t'$ then set $\mathsf{F} \leftarrow \mathsf{true}$; $w_j \leftarrow (c_s \cdot c_\beta)^{\lambda_j \cdot x_{ID_j}}$			
$d_{\beta,j} \leftarrow (d_p/d_{\tilde{p}})^{t_j}$	If $(\hat{d}_{\tilde{p}}/(\hat{c}_{\tilde{p}})^{\hat{x}} = (\hat{h})^p)$ then $d_{\beta,j} \leftarrow (d_p/d_{\tilde{p}})^{t_j}$ else $d_{\beta,j} \xleftarrow{r} G$	G_3	$d_{\beta,j} \xleftarrow{r} G$ G_6
$z_j \leftarrow d_{\beta,j} \cdot (w_j)^{-1}$	If $(c_{\tilde{p}} \in \mathsf{CpSet})$ then $z_j \xleftarrow{r} G$ else $z_j \leftarrow d_{\beta,j} \cdot (w_j)^{-1}$	G_2	
$r_{z_j} \xleftarrow{r} \mathbb{Z}_q, (c_{z_j}, d_{z_j}) \leftarrow (g^{r_{z_j}}, (c_{\tilde{p}})^{r_{z_j}} \cdot z_j),\ \pi_{3j} \leftarrow \mathcal{S}[\mathcal{L}_{S2}^{\mathsf{st}_0, ID_j}](c_{z_j}, d_{z_j}, c_{\tilde{p}}, a_j, d_p/d_{\tilde{p}}, (c_s \cdot c_\beta)^{\lambda_j}),\ \mathsf{Ret}((c_{z_j}, d_{z_j}), \pi_{3j})$			

Figure 4: Games G_0,G_1,...,G_8 used in the security proof of the PPSS protocol

In Figure 4, all games follow the code in boxes that span the width of the figure. However, if a line has two boxes, an unmarked one on the left and one marked "G_i" for some i on the right, then game G_j follows the box on the left for $j < i$ and the box on the right for $j \geq i$. In other words, the boxes on the right mark the differences from G_0 introduced in some game, which are then adhered to in each subsequent game. A special case is the second row of boxes in the code of oracle $\mathsf{Server}_j^{\diamond}$, which contain three boxes: Games $\mathsf{G}_0 - \mathsf{G}_2$ follow the unmarked left-most box, games $\mathsf{G}_3 - \mathsf{G}_5$ follow the middle box marked "G_3", and games $\mathsf{G}_6 - \mathsf{G}_8$ follow the right-most marked "G_6".

Game G_0 portrayed in Figure 4 differs from $\mathcal{A}$'s view of the real protocol in three ways: (1) Figure 4 does not include the interaction corresponding to step **U2** of the $\mathsf{User}^{\diamond}$ oracle. Recall that in the strong security notion the adversary receives a bit indicating whether the User protocol instance accepts or rejects its protocol session. However, in PPSS_2 this bit is determined by whether this User instance receives $t+1$ messages corresponding to server's steps **S1** and **S2** accompanied by proofs $\{\pi_{1j}, \pi_{3j}\}_{j=1}^{t+1}$ which pass the corresponding verification, hence this bit is publicly computable from the adversary's actions; (2) The $\mathsf{User}^{\diamond}$ oracle simulating step **U1** of the User algorithm does not return values λ_j and c_β, but these values are publicly computable from the inputs $\{ID_j, b_j\}_{j=1}^{t+1}$ sent by $\mathcal{A}$ to this User session and from its outputs $\{e_j\}_{j=1}^{t+1}$; (3) In G_0 all oracles output simulated proofs, while in the real interactions these proofs are output by respective prover algorithms on corresponding witnesses.

that it computes $\hat{g} \leftarrow g^{r_0}$ and $\bar{g} \leftarrow A^{r_1}$ for $r_0, r_1 \xleftarrow{r} \mathbb{Z}_q$ and embeds $\hat{y} \leftarrow B$. Then $\mathcal{R}^i_{0,1}$ follows G_0 in all $\mathsf{Server}^\diamond$ calls, but for User sessions it follows G_1 on all sessions prior to the i-th session, and G_0 on all sessions from $i+1$ on, but on the i-th session it sets $\{e_j \leftarrow (\bar{a}_j)^{1/(r_0 r_1)}\}_{j=1}^t$, $c_{\tilde{p}} \leftarrow A^{1/r_0}$, $d_{\tilde{p}} \leftarrow A^{x/r_0} h^p$, $\hat{c}_{\tilde{p}} \leftarrow A$, and $\hat{d}_{\tilde{p}} \leftarrow C^{1/r_0}(\hat{h})^p$. If (A, B, C) is a DDH tuple (g^a, g^b, g^{ab}) then $\mathcal{R}^i_{0,1}(A, B, C) \equiv \mathsf{G}^{i-1}_0$ because for the i-th session the user's output is computed as in G_0, with $r_{\tilde{p}} = a/r_0$ and $\hat{x} = DL(\hat{g}, \hat{y}) = b/r_0$. However if (A, B, C) is a random tuple then $\bar{d}_{\tilde{p}}$ is a random group element, as in G_1, and so $\mathcal{R}^i_{0,1}(A, B, C) \equiv \mathsf{G}^i_0$. It follows that $|p^i_0 - p^{i+1}_0| \leq \epsilon_{ddh} + \epsilon_{SS}$, since we have to add the probability ϵ_{SS} that all proofs $\pi_{1,j}$ verify while some $(a_j, \bar{a}_j)$ sent to the i-th user session is not of the form $(g^{t_j}, (\bar{g})^{t_j})$ for some t_j. □

CLAIM 6. *Games G_1 and G_2 are indistinguishable under DDH. Concretely, $|p_1(s) - p_2(s)| \leq q_U(\epsilon_{ddh} + \epsilon_{SS})$.*

PROOF. We use a hybrid argument over q_U user sessions. For each $i \in [0, q_U]$, we define an intermediate game G^i_1 which follows G_2 except that in $\mathsf{Server}^\diamond$ oracle responses G^i_1 decides whether z_j should be real or random based on whether $c_{\tilde{p}} \in \mathsf{CpSet}[1, i]$, the set of $c_{\tilde{p}}$'s output by the first i $\mathsf{User}^\diamond$ oracle sessions. Note that $\mathsf{G}^0_1 \equiv \mathsf{G}_1$ and $\mathsf{G}^{q_U}_1 \equiv \mathsf{G}_2$. Let $p^i_1 = \Pr[1 \leftarrow (\mathcal{A} \rightleftharpoons \mathsf{G}^i_1)]$. For each $i \in [1, q_U]$ we show reduction $R^i_{1,2}$ which reduces breaking DDH to distinguishing between G^{i-1}_1 and G^i_1. Let the DDH challenge be $(A, B, C) = (g^a, g^b, g^c)$. The reduction follows the $\mathsf{Init}^\diamond$ procedure as in game G_1 except that it picks $r_0, r_1, \hat{x} \xleftarrow{r} \mathbb{Z}_q$ and sets $\hat{g} \leftarrow g^{r_0}$, $\bar{g} \leftarrow A^{r_1}$, and $\hat{y} \leftarrow (\hat{g})^{\hat{x}}$. For the user sessions, the reduction follows the code of G_1 except for i-th session, where it embeds $c_{\tilde{p}} \leftarrow A$ and computes $\hat{c}_{\tilde{p}} \leftarrow (c_{\tilde{p}})^{r_0}$, $d_{\tilde{p}} \leftarrow (c_{\tilde{p}})^x \cdot h^p$, $\hat{d}_{\tilde{p}} \leftarrow (\hat{c}_{\tilde{p}})^{\hat{x}} \cdot (\hat{h})^p$, and $e_j \leftarrow (\bar{a}_j)^{1/r_1}$ for all j. For the server sessions, the reduction $\mathcal{R}^i_{1,2}$ follows the code for G^{i-1}_1 except for the sessions where $c_{\tilde{p}}$ is passed as input; where it computes $(c_{z_j}, d_{z_j}) \leftarrow (B^{r_{z_j}}, C^{r_{z_j}} \cdot z_j)$ where $z_j = d_{\beta,j}/w_j$. Thus reduction $R^i_{1,2}$ hits G^{i-1}_1 on a DDH tuple and it hits G^i_1 on a random tuple. The reduction relies on the correctness of server's messages sent to user's i-th session; but since the event that these are incorrect while servers' proofs verify is bounded by ϵ_{SS}, this yields $|p^i_1 - p^{i+1}_1| \leq \epsilon_{ddh} + \epsilon_{SS}$, and the claim follows. □

CLAIM 7. *Games G_2 and G_3 are indistinguishable under DDH. Concretely, $|p_2(s) - p_3(s)| \leq (n \cdot q_S)(\epsilon_{ddh} + \epsilon_{SS})$.*

PROOF. We use a hybrid argument over nq_S server sessions to argue that G_2 and G_3 are indistinguishable. For each $i \in [0, nq_S]$, we define an intermediate game G^i_2 which follows G_3 on calls to $\mathsf{Server}^\diamond_j$ for $j \leq i$ and follows G_2 on calls to $\mathsf{Server}^\diamond_j$ for $j > i$. Note that $\mathsf{G}^0_2 \equiv \mathsf{G}_2$ and $\mathsf{G}^{nq_S}_2 \equiv \mathsf{G}_3$. Let $p^i_2 = \Pr[1 \leftarrow (\mathcal{A} \rightleftharpoons \mathsf{G}^i_2)]$. For each $i \in [1, nq_S]$ we show reduction $R^i_{2,3}$ which reduces breaking DDH assumption to distinguishing between G^{i-1}_2 and G^i_2. Let the DDH challenge be $(A, B, C) = (g^a, g^b, g^c)$. The reduction follows the $\mathsf{Init}^\diamond$ procedure as in game G_3 except that it sets $(h, \hat{h}, \bar{g}) \leftarrow (B, C^{r_0}, g^{r_1})$ where $r_0, r_1 \xleftarrow{r} \mathbb{Z}_q$. Also, $\mathcal{R}^i_{2,3}$ computes $(a_j, b_j, \bar{a}_j)$ for all $j \in [nq_S]$ as in the protocol, except for $j = i$ where it sets $(a_i, b_i, \bar{a}_i) \leftarrow (A, A^{r_p}, A^{r_1})$. The reduction responds to $\mathsf{Server}^\diamond$ queries for sessions $j < i$ as in G_3, for sessions $j > i$ as in G_2, while on the i-th session it computes $d_{\beta,i} \leftarrow (b_i/e_i)^x \cdot C^p \cdot (\hat{d}_{\tilde{p}}/(\hat{c}_{\tilde{p}})^{\hat{x}})^{-1/r_0}$. If (A, B, C) is a DDH tuple, and the oracle inputs are formed correctly, i.e. proof π_{2i} holds on the correct statement $(a_i, e_i, c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}})$, then $d_{\beta,i} = y^{(r_p - r_{\tilde{p}})t_i} h^{(p-\tilde{p})t_i}$ as in G^{i-1}_2, where $t_i = a$, because in that case $C = h^{t_i}$. Otherwise, if (A, B, C) is a random tuple, then $d_{\beta,j}$ is random as in G^i_2. Since the views are correct except for probability ϵ_{SS}, we get that $|p^i_2 - p^{i-1}_2| \leq \epsilon_{ddh} + \epsilon_{SS}$, and the claim follows. □

CLAIM 8. *For any s, $p^{\neg\mathsf{F}}_3(s) = p^{\neg\mathsf{F}}_4(s)$ and $\Pr[\mathsf{F}_{3,s}] = \Pr[\mathsf{F}_{4,s}]$.*

PROOF. We argue that under condition that event F does not happen the adversarial views in games G_3 and G_4 are identical. This immediately implies that (1) $\Pr[\mathsf{F}_{3,s}] = \Pr[\mathsf{F}_{4,s}]$, and (2) that the conditional probabilities $\Pr[1 \leftarrow (\mathcal{A} \rightleftharpoons \mathsf{G}_3(s)) \mid \neg\mathsf{F}_{3,s}]$ and $\Pr[1 \leftarrow (\mathcal{A} \rightleftharpoons \mathsf{G}_4(s)) \mid \neg\mathsf{F}_{4,s}]$ are the same, and hence the claim follows, because $p^{\neg\mathsf{F}}_i(s) = \Pr[1 \leftarrow (\mathcal{A} \rightleftharpoons \mathsf{G}_i(s)) \mid \neg\mathsf{F}_{i,s}] * \Pr[\mathsf{F}_{i,s}]$ To argue this, note that the only difference between G_3 and G_4 is that in G_3, x_i's are (n, t)-secret-sharing of a random value whereas in G_4 x_i's are (n, t)-secret-sharing of zero. Therefore unless adversary knows $t+1$ shares of x_i's, the view of the adversary in G_3 is identical to its view in G_4. Now, adversary gets to know $t' < t$ shares of x simply by corrupting t' servers. However server queries could possibly leak information about x_i's as in w_j value. But in any server query, if adversary does not use the legitimate password, w_j is masked with a random value in both G_3 and G_4; otherwise if adversary uses the legitimate password, unless event F happens, the number of distinct servers which adversary contacts is bounded by $t - t'$. Thus, unless event F happens, the maximum number of x_i shares that is effectively used in either game is bounded by t. □

CLAIM 9. *Games G_4 and G_5 are indistinguishable under DDH, i.e. $|p^{\neg\mathsf{F}}_4(s) - p^{\neg\mathsf{F}}_5(s)| \leq \epsilon_{ddh}$ and $|\Pr[\mathsf{F}_{4,s}] - \Pr[\mathsf{F}_{5,s}]| \leq \epsilon_{ddh}$.*

PROOF. The proof goes via an easy reduction from DDH which embeds the DDH challenge (A, B, C) by setting $(y, c_s, d_s) \leftarrow (A, B, C \cdot s)$. Note that neither game G_4 nor G_5 needs to know values x and r_s corresponding to (y, c_s, d_s). □

CLAIM 10. *Games G_5 and G_6 are indistinguishable under DDH, i.e. $|p_5(s) - p_6(s)| \leq (n \cdot q_S)(\epsilon_{ddh} + \epsilon_{SS})$ and $|\Pr[\mathsf{F}_{5,s}] - \Pr[\mathsf{F}_{6,s}]| \leq (n \cdot q_S)(\epsilon_{ddh} + \epsilon_{SS})$.*

PROOF. As in the proof of Claim 7, we use a hybrid argument over nq_S server sessions. For $i \in [0, nq_S]$ we define game G^i_5 which follows G_6 on $\mathsf{Server}^\diamond_j$ calls for $j \leq i$ and G_5 on $\mathsf{Server}^\diamond_j$ calls for $j > i$. Note that $\mathsf{G}^0_5 \equiv \mathsf{G}_5$ and $\mathsf{G}^{nq_S}_5 \equiv \mathsf{G}_6$. Let $p^i_5 = \Pr[1 \leftarrow (\mathcal{A} \rightleftharpoons \mathsf{G}^i_5)]$ and let F^i_5 denote event F in G^i_5. For each $i \in [1, nq_S]$ we show a DDH reduction $R^i_{2,3}$ which on input a tuple $(A, B, C) = (g^a, g^b, g^c)$ follows the $\mathsf{Init}^\diamond$ procedure as in game G_5 except that it sets $(y, h, \hat{g}, \hat{h}, barg) \leftarrow (B, g^{r_0}, C^{r_3}, A^{r_1}, g^{r_2})$ where $r_0, r_1, r_2, r_3 \xleftarrow{r} \mathbb{Z}_q$. $\mathcal{R}^i_{2,3}$ computes all $(a_j, b_j, \bar{a}_j)$ as in the protocol except for $j = i$ where it sets $(a_i, b_i, \bar{a}_i) \leftarrow (A, A^{r_p}, A^{r_2})$. The reduction responds to $\mathsf{Server}^\diamond$ queries for sessions $j < i$ as in G_6, for sessions $j > i$ as in G_5, while on the i-th session it computes $d_{\beta,i} \leftarrow C^{r_p} \cdot A^{p \cdot r_0} \cdot (\hat{d}_{\tilde{p}}/(\hat{c}_{\tilde{p}})^{\hat{x}})^{-r_0/r_1} \cdot (\hat{c}_{\tilde{p}})^{-1/r_3}$ If (A, B, C) is a DDH tuple, and proof π_{2i} holds on the correct statement $(a_i, e_i, c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}})$, then $d_{\beta,i} = g^{x(r_p - r_{\tilde{p}})t_i} h^{(p-\tilde{p})t_i}$ as in G^{i-1}_5, where $t_i = a$ and $x = b$, because in that case $C = g^{x \cdot t_i}$. Otherwise $d_{\beta,j}$ is random, as in G^i_5. Since the views are correct except for probability ϵ_{SS}, we get both $|p^i_5 - p^{i-1}_5| \leq \epsilon_{ddh} + \epsilon_{SS}$ and $|\Pr[\mathsf{F}^i_5] - \Pr[\mathsf{F}^{i-1}_5]| \leq \epsilon_{ddh} + \epsilon_{SS}$, and the claim follows. □

CLAIM 11. *Games G_6 and G_7 are indistinguishable under DDH, i.e. $|\Pr[\mathsf{F}_{6,s}] - \Pr[\mathsf{F}_{7,s}]| \leq q_U(\epsilon_{ddh} + \epsilon_{SS})$.*

PROOF. The proof goes via a hybrid reduction from DDH over the $\mathsf{User}^\diamond$ oracle sessions, for $i = 1, .., q_U$, and it is very similar to the reduction in the proof of Claim 5. The reduction picks $(\hat{g}, \bar{g}) \leftarrow (g^{r_0}, g^{r_1})$ for random r_0, r_1 and $\hat{y} = (\hat{g})^{\hat{x}}$ for random $\hat{x}$, and embeds the DDH challenge (A, B, C) by setting $y \leftarrow B$, and then in the i-th $\mathsf{User}^\diamond$ session it assigns $c_{\tilde{p}} \leftarrow A$, $d_{\tilde{p}} \leftarrow C \cdot h^p$, $\hat{c}_{\tilde{p}} \leftarrow A^{r_0}$, $\hat{d}_{\tilde{p}} \leftarrow (\hat{c}_{\tilde{p}})^{\hat{x}}(\hat{h})^p$, and sets each e_j as $e_j \leftarrow (\bar{a}_j)^{1/r_0}$. □

$\underline{\mathsf{Init}(p)}$ (on public parameters κ, n, t)

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{EKg}$, $\{(\mathsf{ssk}_i, \mathsf{vk}_i) \leftarrow \mathsf{SKg}\}_{i=1}^n$,
$(\tilde{\mathsf{st}_0}, \tilde{\mathsf{st}_0}, ..., \tilde{\mathsf{st}_n}) \leftarrow \mathsf{PPSS.Init}(p, \mathsf{sk})$
$\mathsf{st}_0 \leftarrow (\tilde{\mathsf{st}_0}, \mathsf{pk}, \{\mathsf{vk}_i\}_{i=0}^n)$, $\{\mathsf{st}_i \leftarrow (\tilde{\mathsf{st}_i}, \mathsf{ssk}_i)\}_{i=1}^n$

$\underline{\mathsf{Client}(\mathsf{st}_0, \tilde{p}, uId) \rightleftharpoons (\mathsf{Server}(\mathsf{st}_0, \mathsf{st}_1), \cdots, \mathsf{Server}(\mathsf{st}_0, \mathsf{st}_n))}$

C1 (Client) Pick $sId \xleftarrow{r} \{0,1\}^\kappa$;
Send (uId, sId) to each Server;
Run $\mathsf{PPSS.User}(\tilde{st_0}, \tilde{p})$ with the Servers.

S1 (Server_j) Run $\mathsf{PPSS.Server}_j(\tilde{\mathsf{st}_0}, \tilde{\mathsf{st}_j})$ with the Client;
On abort, assign $k_j \leftarrow \perp$ on session sId and stop.
Otherwise pick $k_j \xleftarrow{r} \{0,1\}^\kappa$;
Compute $\sigma_j \leftarrow \mathsf{Sign}(\mathsf{ssk}_{Id_j}, (uId, sId, k_j))$
Compute $e_j \leftarrow \mathsf{Enc}(\mathsf{pk}, (k_j, \sigma_j))$;
Send e_j to Client.

C2 (Client) Let $\hat{sk}$ be the output of the PPSS instance.
If $\hat{sk} = \perp$ then set $k_j \leftarrow \perp$ for all j and stop.
Let $(\hat{k}_j, \hat{\sigma}_j) \leftarrow \mathsf{Dec}(\hat{\mathsf{sk}}, e_j)$ for all Server_j;
For each j, if $\mathsf{Vrfy}(\mathsf{vk}_{Id_j}, \hat{\sigma_j}, (uId, sId, \hat{k_j})) = 1$
then set $k_j \leftarrow \hat{k_j}$ on session sId with Server_j;
o/w set $k_j \leftarrow \perp$ on this session.

Figure 5: TPAKE from PPSS

CLAIM 12. *Games* G_7 *and* G_8 *are indistinguishable under DDH, e.g.* $|\Pr[\mathsf{F}_{7,s}] - \Pr[\mathsf{F}_{8,s}]| \leq \epsilon_{ddh}$.

PROOF. The proof goes via an easy reduction from DDH which embeds the DDH challenge (A, B, C) by setting $(y, c_p, d_p) \leftarrow (A, B, C \cdot h^p)$. Note that neither game G_7 nor G_8 needs to know values x and r_p corresponding to (y, c_p, d_p). □

4. EFFICIENT T-PAKE FROM PPSS

A password protected secret sharing (PPSS) scheme can be used as a black box to achieve a threshold password authenticated key exchange (T-PAKE) protocol (in the public key model) at very little additional cost. In particular, the round complexity of the resulting T-PAKE is the same as the PPSS because all the T-PAKE messages can be piggybacked onto the PPSS protocol flows. Figure 5 shows a secure T-PAKE protocol assuming that $\mathcal{E} = (\mathsf{EKg}, \mathsf{Enc}, \mathsf{Dec})$ is a chosen ciphertext attack secure public key encryption, $\mathcal{S} = (\mathsf{SKg}, \mathsf{Sign}, \mathsf{Vrfy})$ is a signature scheme which is existentially unforgeable under chosen message attack, and PPSS is a strongly secure password protected secret sharing protocol. For lack of space we omit the formal proof that this construction satisfies T-PAKE security, but, very briefly, the signatures and CCA encryption scheme ensure that the network adversary cannot re-route messages from a session in which honest players are involved, or modify them in any way, and hence in particular all User sessions are independent of each other. Then by the security of the PPSS scheme, except for $\lfloor q_S/(t - t' + 1) \rfloor \cdot (1/|D|)$ probability, the view of the PPSS protocol initialized with the real decryption key sk is indistinguishable from a view where sk is replaced by an independent key, in which case CCA security of encryption ensures that $\mathcal{A}$ gets no information about any unrevealed session keys even given a capability to reveal any other session keys (handled by decryption queries in a reduction to CCA encryption security).

5. REFERENCES

[1] M. Abdalla, M. Bellare, and P. Rogaway. The oracle diffie-hellman assumptions and an analysis of DHIES. In D. Naccache, editor, *CT-RSA*, volume 2020 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 2001.

[2] M. Bellare, J. A. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In *EUROCRYPT*, pages 236–255, 1998.

[3] J. Brainard, A. Juels, B. Kaliski, and M. Szydlo. Nightingale: A new two-server approach for authentication with short secrets. In *12th USENIX Security Symp*, pages 201–213. IEEE Comp. Soc. 2003.

[4] J. Camenisch and M. Michels. Proving in Zero-Knowledge that a Number Is the Product of Two Safe Primes. In *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 107–122, 1999.

[5] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive security for threshold cryptosystems. In *CRYPTO'99*, volume 1666 of *LNCS*, pages 98–115, 1999.

[6] Y. Desmedt and Y. Frankel. Threshold Cryptosystems. In *CRYPTO '89*, volume 435 of *LNCS*, pages 307–315, 1990.

[7] M. Di Raimondo and R. Gennaro. Provably secure threshold password-authenticated key exchange. *J. Comput. Syst. Sci.*, 72(6):978–1001, 2006.

[8] Y. Dodis, M. K. Franklin, J. Katz, A. Miyaji, and M. Yung. Intrusion-resilient public-key encryption. In *CT-RSA*, pages 19–32, 2003.

[9] Y. Dodis, J. Katz, S. Xu, and M. Yung. Key-insulated public key cryptosystems. In L. R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 65–82. Springer, 2002.

[10] M. Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *Crypto'05*, 2005.

[11] W. Ford and B. S. K. Jr. Server-assisted generation of a strong secret from a password. In *WETICE*, pages 176–180, 2000.

[12] J. A. Garay, P. D. MacKenzie, and K. Yang. Strengthening zero-knowledge protocols using signatures. In E. Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2003.

[13] S. Goldwasser, S. Micali, and R. L. Rivest. A "paradoxical" solution to the signature problem. In *IEEE Annual Symposium of Foundations of Computer Science (FOCS'84)*, pages 441–448, 1984.

[14] D. Jablon. Password authentication using multiple servers. In *CT-RSA'01: RSA Cryptographers' Track*, pages 344–360. Springer-Verlag, 2001.

[15] J. Katz, P. Mackenzie, G. Taban, and V. Gligor. Two-server password-only authenticated key exchange. In *Proc. Applied Cryptography and Network Security ACNSAÂŠ05*, 2005.

[16] J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques*, 2001.

[17] LastPass. Lastpass password manager, 2009. Available at `https://lastpass.com`.

[18] P. D. MacKenzie and M. K. Reiter. Networked cryptographic devices resilient to capture. *Int. J. Inf. Sec.*, 2(1):1–20, 2003.

[19] P. D. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. *J. Cryptology*, 19(1):27–66, 2006.

[20] Mozilla Labs. Weave sync, 2009. Available at `http://labs.mozilla.com/projects/weave`.

[21] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciph. attacks. In *STOC*, pages 427–437. ACM, 1990.

[22] A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, pages 543–553, 1999.

[23] A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, Nov. 1979.

[24] V. Shoup. Practical Threshold Signatures. In *EUROCRYPT'00*, volume 1807 of *LNCS*, pages 207–220, 2000.

[25] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *J. Cryptology*, 15(2):75–96, 2002.

[26] S. Xu and R. S. Sandhu. Two efficient and provably secure schemes for server-assisted threshold signatures. In *CT-RSA*, 2003.