Securing Password Recovery through Dispersion

Sushil Jajodia
Center for Secure Information Systems
George Mason University
Fairfax, Virginia, USA
jajodia@gmu.edu

Witold Litwin
Lamsade
Université Paris Dauphine
Paris, France
wlitwin@dauphine.fr

Thomas Schwarz, S.J.
Informática y Ciencias de la Computación
Universidad Católica del Uruguay
Montevideo, Uruguay
tschwarz@ucu.edu.uy

Abstract—**Passwords form the Achilles heel of most uses of modern cryptography. Key recovery is necessary to provide continuous access to documents and other electronic assets in spite of possible loss of a password. Key escrow services provide key recovery for the owner, but need to be trusted. Additionally, a user might want to divulge passwords in case of his/her death or incapacitation, but not before.**

We present here a scheme that uses dispersion to provide trusted escrow services. Our scheme uses secret sharing to disperse password recovery information over several escrow services that authenticate based on a weak password. To protect against dictionary attacks, each authentication attempt takes a noticeable, but tolerable time (e.g. minutes). We achieve this by having the share of the secret be the solution of a puzzle that is solved by brute force in time depending on the number of processors employed. This additionally prevents escrow agencies from optimizing their part in recovering a password by pre-computing and storing their share in a more accessible and hence vulnerable format.

Keywords-**Password Escrow, Password Recovery, Dispersion, Cloud**

I. Introduction

Cryptography solves many security problems, but key management remains a difficult problem. To prevent loss of a crucial key, a user can entrust the key to an escrow agency, but if she does so, then she has to trust the escrow service. The escrow service needs to prevent accidental and malicious disclosure by insiders and outsiders, but additionally convince the user that the measures taken are of sufficient strength.

We present here a scheme that allows a user to recover her passwords from a set of escrow services without having to trust any one of them. She stores her passwords in a password file, protected with a strong key. The key is backed up “in the cloud” at several services and protected with a weak, but easy to be remembered *master password*. We use a standard secret sharing scheme to divide the key among the services. For recovery, the user sends her weak password to all the servers who after a while respond with their share of the strong password. The user regains her key from these shares. Our contribution lies in the imposition of guaranteed work for the recovery of a share that prevents dictionary attacks by an adversary. Using a weak password allows the user to distribute hints allowing her friends and family to guess the weak password and recover her password file key.

Our scheme functions even if a proportion is not operational and even if a smaller portion returns false information. Additionally, no site can tell if the user has authenticated successfully. The nature of the scheme prevents a participant escrow service from ”pre-computing” its share of the private key and thus exposing it to additional attacks. The heavy use of computational resources needed for regenerating the share leaves an auditable trace.

The use of a weak password to protect a strong password presents an unusual trade-off between usability and security, where security is weakened by design. While there are certain passwords that should never be subjected to such a scheme (some types of bank accounts come to mind), the vast majority of my passwords at least should be accessible to friends and colleagues in case of temporary disability or death. In a certain sense, the mandatory key escrow once discussed in the USA, presents a similar trade-off, where security was to be absolute to protect against private adversaries, but not against law enforcement agencies provided with a warrant.

II. Related Work

Key escrow mandated by government was a hotly contested issue in the nineties in the United States and much work has been devoted to define the legal, ethical, and technical issues and to design, prototype, and standardize key recovery mechanisms, as the work by Bellare and Goldwasser [2], the work on the Clipper proposal by US government [16], the proposal by Verheul and van Tilborg [27], and the risk evaluation by Abelson and colleagues [1] on the technical side, and the ethical and legal assessments by Denning and Baugh [7] and Singhal [25] among many others show.

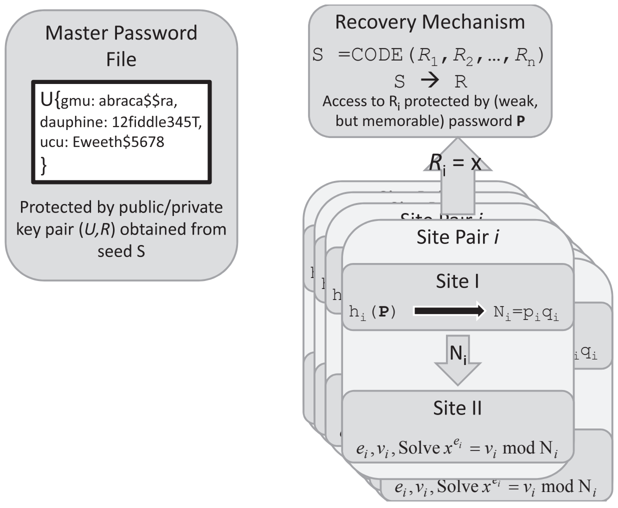
Our goal here is quite different from that of mandatory escrow system, which has to treat the user as a potential adversary who might not want to put his key in escrow. In mandatory systems, the agency can only – if ever – read the conversation *after* it has happened. When accepting a key or observing a conversation, the agency has to decide whether she will be able to read the conversation sometimes in the future, when it might be legal to do so. Super-encryption, in which the conversation is encrypted twice, once with a key kept from the escrow authority and then with a key given to her, thwarts simple schemes.

In common is the user's desire to control *when* the escrow agency can commence recovering a key. We want to prevent *early recovery* in favor of *delayed recovery* [2]. This goal is similar to Rivest's and Shamir's timed release crypto [23], where a certain amount of computation needs to be performed to obtain a secret.

The advent of cloud services has put distributed computing at the fingertips of average users and web services allow easy interaction as intermediaries between users and cloud services. This development allows us to use highly distributed applications in order to protect user data from disclosure and alteration. The keys can be stored within the distributed system [14], [24] or can be protected within the system acting as an escrow system, as was done in our previous work [12], [13]. We pursue the same line of research here, but replace user authentication to sites with a weak password as credential.

Our scheme uses secret sharing to let the user distribute key information to various services. Any such solution needs to survive accidental unavailabilities and some malicious activities at the services [9]. *Verifiable Secret Sharing* (VSS) guarantees (P1) that a significant fraction of the shares are needed to recover the secret, but that (P2) the recovery is still possible if a significant fracion of the shares is corrupted [10]. In our context, we do not need a third property of VSS, namely (P3) that the recipients of the shares can check that the sharing has been performed correctly. The VSS protocol by Chor, Goldwasser, Micali, and Averbuch [6] and its many successors, see [18], [21], have to spend considerable effort on implementing it. To prevent *cheaters* from sabotaging the reconstruction of the secret by presenting false shares, one can simply have the distributor sign the secret [11]. In our case, the user would have to be able to access the data necessary to confirm her own signature, (e.g. her own public key), so that this solution is not easy. However, Tompa and Woll [26] were the first one to achieve (P1) and (P2) in the case of Shamir's secret sharing algorithm with only a small probability of success by a band of cheaters. The problem has received a large amount of atention, and solutions have been extended to general threshold schemes, such as the work by Ogata, Kurosawa, and Stinson [19] and Obana [18] among many others.

We present our solution in terms of a linear erasure correcting code using with a standard ramp scheme, following in the footsteps of McEliece and Sarwate [17]. The reasons for our choice of presentation are solely the simplicity and robustness of linear codes and the lack of absolute security in our proposal. A more involved scheme depends on much better preparation to allow its use for many years in the



Master Password File
U{gmu: abraca$$ra, dauphine: 12fiddle345T, ucu: Eweeth$5678 }
Protected by public/private key pair (U,R) obtained from seed S
Recovery Mechanism
S =CODE(R1, R2, …, Rn)
S → R
Access to Ri protected by (weak, but memorable) password P
Ri = x
Site Pair i
Site I
hi(P) → Ni=piqi
Ni
Site II
ei, vi, Solve x^ei = vi mod Ni

Figure 1: Password Retrieval System Overview

future. Imagine only an implementation that needs to use Python 1.4 in about ten years.

As we distribute a private key into shares, we need to worry about the impact of revealing bits of the keys, as is shown in the work of Blömer, Ernst, May and colleagues [4], [8] among many others. Our solution involves using a good, but standard random number generator in order to re-generate the public-private key pair from the secret obtained from the servers.

III. Trust through Dispersion

In our scheme, a user uses several web-based services, each of which has access to on-demand cloud resources. The services can be generic cloud servers, that execute a user-owned program, or they can be commercial solutions that provide services according to our scheme. We assume that the actual services belong to different administrative domains and that the user is inclined to trust them.

We first give an overview of our system (Figure 1). The user maintains a password file that is protected using asymmetric cryptography. The public key is used to encrypt the contents and is stored with the file so that the file can be updated. The file itself can be stored in multiple copies using web-based storage. Our distributed retrieval system recovers the private key (needed to read the password file) using a weak *master password*. If the user submits the master password to all web services, then he receives enough shares in order to reconstruct the private key *R* and gain the capacity to retrieve the password file. To protect against rogue sites, the reconstruction of *R* uses an error correcting code. Each obtain a piece of information on *R*, a large computational effort is required that necessitates the use of cloud services and is large enough to generate traces such as billing for services. It also prevents dictionary attacks on the master password. While the user does not authenticate herself to each service, she is informed by correctly functioning sites of each retrieval effort, giving her months to counteract any effort at guessing the master password by an adversary.

A. Retrieval System Operations

A user starts out by creating a public-private key pair (U,R) using a publicly available random number generator RG and a seed S. She sets up a file with authentication information for every important account the user has. To maintain access to this file, called the *password file*, the user stores the public key U and the file in several back-up copies, distributed to a number of sites (”in the cloud”). To prevent misuse such as adding to or substituting the password file, the password file needs to be integrity protected.

The user does not maintain the private key R. Given RG, the random number generator, and S, she can recreate the public-private key pair on demand. She stores S in an indirect, distributed way on various sites “in the cloud”. The access to S is protected by a *master password* P. This password is maintained in a large list of possible passwords. In contrast to the other passwords, P can be a weak password, thus as a minimally changed word from a common-language dictionary.

The user first break the seed needed to generate the private key R into various components R_1, R_2, $\dots R_n$ using an error correcting code that can correct k errors and $2k+1$ erasures, or any corresponding combination of errors and erasures. For example, the user can use a standard, linear, maximum distance separable block code with length n and $2k+1$ parity symbols [15]. The seed S needed to obtain the private key R could be a single data symbol for the code or could be a recovered as a combination of data symbols. An alternative uses secret splitting or a ramp scheme, in which the key is split into n pieces such that k are necessary to reassemble the code. We can use a threshold scheme with protection against cheaters, for example by signing the shares with a different private-public key pair and storing the public key with the share. A cheating site can of course generate another private-public key, but a majority of cheating sites need to collaborate in order to convince the user that the replacement assymmetric key is the correct one and not the one originally used.

Each component R_i is stored in indirect form at a site. Each site runs an application that can respond to an invocation by the user. To set up the site, the user generates a *puzzle*, that depends on a user input, and whose solution is a component R_i if the user input is the master password P. The solution of the puzzle takes a certain amount of computation (in expectation). Furthermore, a site should not be able to decide whether a master password might or might not be correct. A puzzle that can be completed in an adequate time (minutes, but neither days nor seconds) can usually not solve for a complete key component R_i. We therefore assume a *decomposition function* f, which need not be hard to compute or invert. The user uses this function to break R_i into two components, $R_{i,1}$ and $R_{i,2}$ where $R_{i,1}$ has an information content of b bites. Thus, $R_i = f(R_{i,1}, R_{i,2})$. One component, $R_{i,2}$, is stored at the site. The other one is the solution of a puzzle Z_i that depends on the master password. The puzzle always has a solution, whether or not the correct value of P has been provided.

To retrieve R and thus (re-)gain access to the password file, the user submits a hash of the master password P at all sites. Each site uses the master password to recover $R_{i,1}$, then combines it with the (locally available) other component $R_{i,2}$ to obtain R_i and send this value to the user. After waiting for a certain amount of time for the results to come in, the user uses the error correcting code to reconstruct R. As R is the private key to the password file, the user now has regained access to it.

An adversary without access to any site can mount a dictionary attack on P. The adversary guesses a value P' and sends it to all sites. The sites will give the adversary components R_i', which the adversary can use to calculate a value R'. Only by trying out R' on the password file, can the adversary decide whether he guessed P correctly. Each guess takes a certain amount of time to verify. If sites do not permit parallel searches (which is not attractive since they have to use their alotted time slots to calculate one solution) and if they send a notification to the user about each attempt, then this brute force attack is unlikely to succeed. The adversary has a small window of time before the user knows that someone tries to steal her credentials. A typical dictionary attack with at least a minute lag between submitting a guess for the weak master password would take a week to process half of a small dictionary.

An adversary who has the information stored at a single site can at best mount a partial dictionary attack that reveals values R_i' in dependence of all possible master passwords P'. There is no sufficient information in order to decide which of those values are useful. In any case, if the decomposition of R into components R_i is sufficiently random (see below), then knowledge of a single value is not sufficient. Only an adversary who controls information at several sites can gather enough components R_i' to calculate a corresponding private key R', but would still have to verify this by getting hold on the password file.

B. Puzzle Construction

A puzzle is a problem that depends on the input (in our case, the hash of P), and that can always be solved using a certain amount of time. It always returns a result if the input appears formally correct. Thus, a single execution of the puzzle does not reveal whether the input is the hash of the correct password or a fake value. Furthermore, it should be impossible for a site to preprocess a puzzle, even though the site might store some information.

Unfortunately, we were not able to think of a class of puzzles that fulfills these requirements. Instead, we implement them using two sites for each puzzle. Our solution uses the RSA public key system. The RSA *generator* takes as input a pseudo-random number and generates a number N and two exponents e and d that define functions $f: \mathbb{Z}_N \to \mathbb{Z}_N$, $x \mapsto x^e$ and $f^{-1}: \mathbb{Z}_N \to \mathbb{Z}_N$, $x \mapsto x^d$. The exponent e can be freely chosen, subject only to the condition that g.c.d.$(e, \phi(N)) = 1$ where ϕ is Euler's function. The RSA generator yields a trapdoor permutation. The data given to another entity is (N, e) and the trapdoor information is (N, d) or a decomposition of N as a product of two prime numbers of approximately the same magnitude, $N = p \cdot q$. Following Bellare and Rogaway [3], we call an inverting algorithm a t-inverter (for a function $t: \mathbb{N} \to \mathbb{N}$) if the inverter's running time on input of k bits of N is bounded by $t(k)$. An inverting algorithm I (t, ε)-*breaks* RSA if for each input size k, the success probability in inverting within time $t(k)$ is at least ε. Here, we allow ε to be a function of k, i.e. $\varepsilon: (N) \to [0,1]$, $k \mapsto \varepsilon(k)$.

The fastest algorithms for factoring large composite numbers belong still to the special Number Field Sieve family [5], [20], [22] and have a complexity of

$$\exp((C + \sqrt[3]{})64/9(k^{1/3}(\log(k))^{2/3}$$

One [3] can therefore conclude that RSA is (t, ε) secure if (t, ε) satisfy

$$t(k)/\varepsilon(k) \leq C \exp(k^{1/4})$$

We generate and solve the puzzle in two different sites. The user sends a hash of his password to the first site, Site I. Site I stores a number k of bits. The site uses the user-submitted password hash to generate two prime numbers p and q of approximately $k/2$ bits in a deterministic manner, calculates their product N and sends N to the second site, site II. Site II has a value v stored and needs to solve the puzzle

$$x^e \equiv v \pmod{N}$$

for the unknown x. Site II can use a distributed factorization algorithm or it can use a distributed brute force attack guessing directly x. Factorization algorithms have components that can be easily parallelized [22]. Site II returns the solution of the puzzle to the user. By construction, the puzzle always has a solution, whatever the password hash submitted by the user and hence input N submitted to Site II might be.

To create the data stored at the sites, the user needs first to decide on a sufficiently large number of bits k for each of the second sites, taking into account the current level of factoring algorithms and the feasibility of a brute force attack. The user then uses a given implementation A to generate the k-bit number N that is the product of two primes p and q. The user stores this implementation and her choice of k at each Site I. The user uses a standard hash function h, salted with the address of each Site I to generate pairs (p_i, q_i) for each site I, calculated from $h(i, P)$. The user then selects an exponent e_i and calculates $v_i = R_i^{e_i} \pmod{p_i \cdot q_i}$. The user sends A and k to each site I. She sends e_i and v_i to each corresponding Site II.

A retrieval operation takes a possible password P', calculates $h(i, P')$, and sends the result to each of the n sites I. Each site I generates N_i, based on the previously stored algorithm A and k, and sends the result – uniquely determined by P' – to the corresponding site II. The site II has stored v_i and e_i and solves the puzzle $x^e \equiv v \pmod{N}$. The site returns the result to the user. The user obtains in this way n solutions $R'_1, R'_2, \ldots, R'_n$. Using the error correcting code, she calculates R'. This code can survive a combination of normal and byzantine failures because of its error and erasure correcting aspect.

IV. Vulnerability Analysis

We first assume that all sites behave benignly, by which we understand the possibility of byzantine failure, but no direct malicious attempts. In this case, retrieval is successful if a majority of the sites function correctly because of the use of the erasure-correcting code.

We next consider the case of an adversary without access to any sites. We assume that the adversary can break any potential authentication procedure of the user at each site. We also assume that an adversary can gain access to a copy of the password file. Since the password file is stored with the public key U used to encrypt it, the adversary can update the file with additional information or can replace the file with a complete copy. The integrity of the password file needs to be protected, for example by adding a Message Authentication Code (MAC). The MAC can use the weak master password as a key without danger, since the whole contents of the file, including the MAC, are encrypted.

To obtain the (weak) master password, the adversary can mount a guessing attack. If the user has instructed all sites to send her email or twitter alarms whenever the retrieval function is invoked, then repeated guessing attacks can be stalled by securing the master file of passwords. If we allow the user to stop the services, then we expose her to a denial-of-service attack. If the guessing attack is successful, then the adversary still only has the private key to read the master password file.

An adversary with sniffing capability or an insider at a malicious site can use traffic analysis to recognize repeated submission of the same password hashes. This information is sufficient to identify the correct password hash and mount a dictionary attack, assuming of course that the hash function is publicly known.

An adversary who has gained access to a Site I can intercept the a retrieval attempt by the original user and generate the local value of $N_i = p_i \cdot q_i$. Without knowing the corresponding value of v_i, he cannot use his knowledge. We also note that a Site I cannot generate N_i without a previous retrieval attempt.

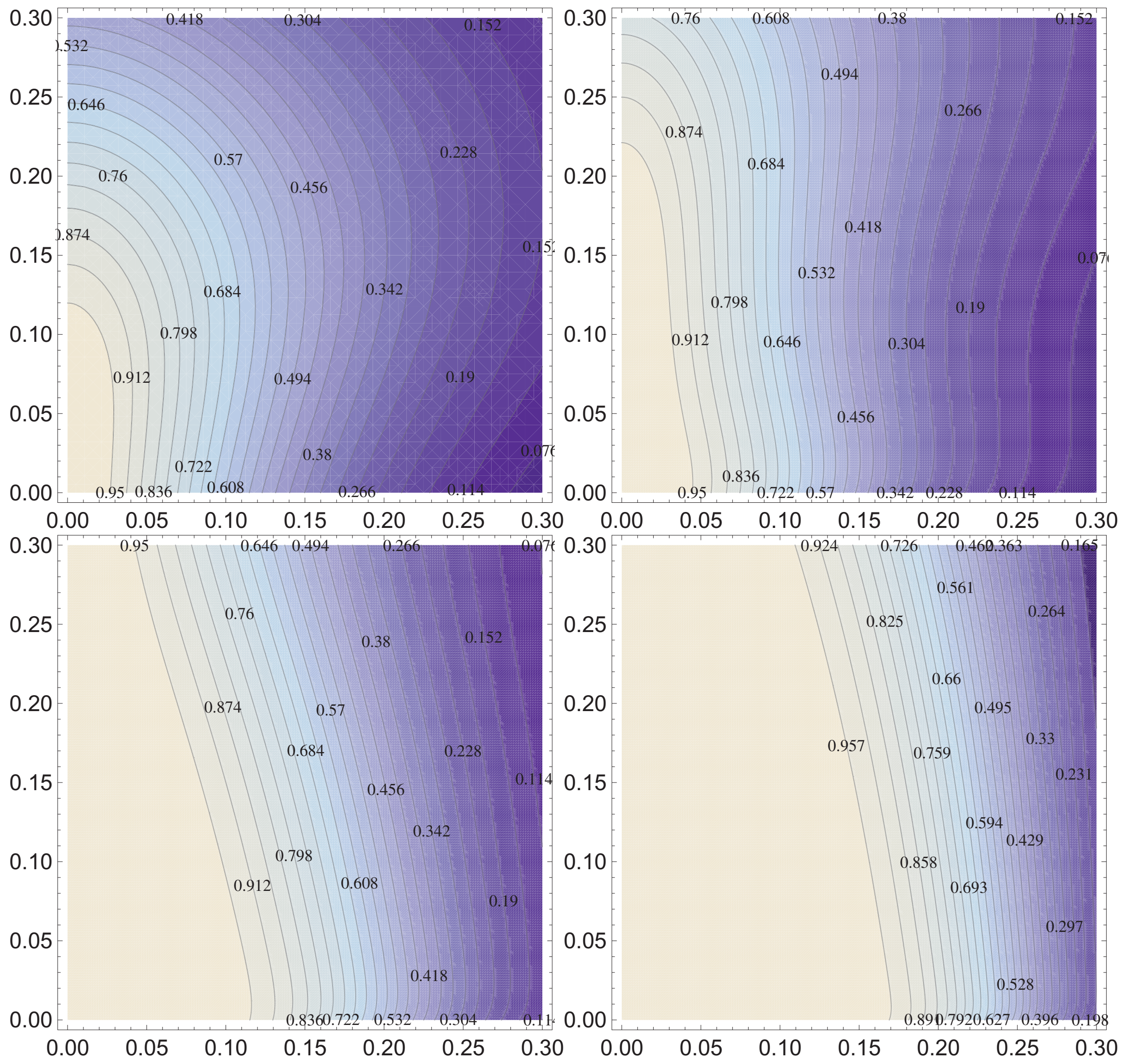


Figure 2: Contour graphs for the probability of successful retrieval using a 4 out of 7 (top-left), a 4 out of 10 (top-right), a 4 out of 20, and a 4 out of 40 scheme. The x-axis gives the probability of malicious failure and the y-axis the probability of simple failure.

An administrator of a Site II only receives a value N_i if there is a retrieval attempt. Thus, there is no possibility of preprocessing without generating the factoring of many different products N'. He will have to generate all possible prime numbers and store their products to precompute the solutions of the puzzle.

An adversary with access to a pair of corresponding Sites I and Sites II can easily retrieve R_i. Unfortunately for the adversary, he will have to retrieve more than one of these values and in addition the master password file.

If we use a verifiable secret sharing scheme, the odds for the adversary are low. In such a scheme, we can (at least with very high probability) detect a falsified share so that the adversary might just as well incite the site not to return a value at all. The costs are more logistic as we need to store a description of the scheme with each server in a manner that can still be read in the reasonably far future.

V. Conclusion

We have presented a key retrieval system that allows the user to add site-key combinations to the protected keys and more importantly, to recover access to this information using a weak password. Our system combines the advantages of a weak password (memorable, easily transmittable) with protection against brute force attacks. Our scheme uses different cloud-based services, but does not depend on the security of authentication to these services for its security.

References

[1] H. Abelson, R. Anderson, S. Bellovin, J. Benaloh, M. Blaze, W. Diffie, J. Gilmore, P. Neumann, R. Rivest, J. Schiller,

and B. Schneier, “The risks of key recovery, key escrow, and trusted third-party encryption,” *World Wide Web Journal*, vol. 2, no. 3, pp. 241–257, 1997.

[2] M. Bellare and S. Goldwasser, “Verifiable partial key escrow,” in *Proceedings of the 4th ACM Conference on Computer and Communications Security*. ACM, 1997, pp. 78–91.

[3] M. Bellare and P. Rogaway, “The exact security of digital signatures-how to sign with RSA and Rabin,” in *Advances in Cryptology-Eurocrypt96*. Springer, 1996, pp. 399–416.

[4] J. Blömer and A. May, “New partial key exposure attacks on rsa,” *Advances in Cryptology-CRYPTO 2003*, pp. 27–43, 2003.

[5] G. Childers, “Factorization of a 1061-bit number by the special number field sieve,” http://eprint.iacr.org, 2012.

[6] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, “Verifiable secret sharing and achieving simultaneity in the presence of faults,” in *26th Annual Symposium on Foundations of Computer Science*. IEEE, 1985, pp. 383–395.

[7] D. Denning and W. Baugh Jr, “Key escrow encryption policies and technologies,” *Villanova Law Review*, vol. 41, p. 289, 1996.

[8] M. Ernst, E. Jochemsz, A. May, and B. De Weger, “Partial key exposure attacks on rsa up to full size exponents,” *Advances in Cryptology–EUROCRYPT 2005*, pp. 555–555, 2005.

[9] G. Ganger, P. Khosla, M. Bakkaloglu, M. Bigrigg, G. Goodson, S. Oguz, V. Pandurangan, C. Soules, J. Strunk, and J. Wylie, “Survivable storage systems,” in *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX’01. Proceedings*, vol. 2. IEEE, 2001, pp. 184–195.

[10] R. Gennaro, “Theory and practice of verifiable secret sharing,” Ph.D. dissertation, Massachusets Institute of Technology, 1996.

[11] L. Guillou and J. Quisquater, “A ”paradoxical” indentity-based signature scheme resulting from zero-knowledge,” in *Advances in Cryptology – Crypto88*. Springer, 1990, pp. 216–231.

[12] S. Jajodia, W. Litwin, and T. Schwarz, “Privacy of data outsourced to a cloud for selected readers through client-side encryption,” in *Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2011.

[13] ——, “Recoverable encryption through noised secret over large cloud,” in *5th International Conference on Data Management in Cloud, Grid and P2P Systems (Globe 2012)*. LNCS-Springer Verlag, 2012.

[14] ——, “LH*RE: A scalable distributed data structure with recoverable encryption,” in *CLOUD ’10: Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 354–361.

[15] F. MacWilliams and N. Sloane, *The theory of error-correcting codes*. North-Holland, 2006.

[16] B. McConnell and E. Appel, “Enabling privacy, commerce, security and public safety in the global information infrastructure,” *Washington, DD: Office of Management and Budget, Interagency Working Group on Cryptography Policy*, 1996.

[17] R. McEliece and D. Sarwate, “On sharing secrets and reed-solomon codes,” *Communications of the ACM*, vol. 24, no. 9, pp. 583–584, 1981.

[18] S. Obana, “Almost optimum t-cheater identifiable secret sharing schemes,” *Advances in Cryptology–EUROCRYPT 2011*, pp. 284–302, 2011.

[19] W. Ogata, K. Kurosawa, and D. Stinson, “Optimum secret sharing scheme secure against cheating,” *SIAM Journal on Discrete Mathematics*, vol. 20, no. 1, pp. 79–95, 2006.

[20] J. Papadopoulos, “Msieve, 2012,” Project site: http://msieve.sourceforge.net.

[21] T. Pedersen, “Non-interactive and information-theoretic secure verifiable secret sharing,” in *Advances in Cryptology–CRYPTO91*. Springer, 1992, pp. 129–140.

[22] I. Popovyan, “Efficient parallelization of Lanczos type algorithms,” http://eprint.iacr.org, 2011.

[23] R. L. Rivest, A. Shamir, and D. A. Wagner, “Time-lock puzzles and timed-release crypto,” Massachusetts Institute of Technology, Cambridge, MA, USA, Tech. Rep., 1996.

[24] T. Schwarz and D. Long, “Clasas: a key-store for the cloud,” in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 267–276.

[25] A. Singhal, “Piracy of privacy-a fourth amendment analysis of key escrow cryptography, the,” *Stanford Law and Policy Review*, vol. 7, p. 189, 1995.

[26] M. Tompa and H. Woll, “How to share a secret with cheaters,” *journal of Cryptology*, vol. 1, no. 3, pp. 133–138, 1989.

[27] E. Verheul and H. van Tilborg, “Binding ElGamal: a fraud-detectable alternative to key-escrow proposals,” in *Advances in Cryptology, EUROCRYPT97*. Springer, 1997, pp. 119–133.