Keeping Data Secret under Full Compromise using Porter Devices

Christina Pöpper
System Security Group
Computer Science
ETH Zurich
poepperc@inf.ethz.ch

David Basin
Information Security Group
Computer Science
ETH Zurich
basin@inf.ethz.ch

Srdjan Čapkun
System Security Group
Computer Science
ETH Zurich
capkuns@inf.ethz.ch

Cas Cremers
Information Security Group
Computer Science
ETH Zurich
cremers@inf.ethz.ch

ABSTRACT

We address the problem of confidentiality in scenarios where the attacker is not only able to observe the communication between principals, but can also fully compromise the communicating parties (their devices, not only their long term secrets) after the confidential data has been exchanged. We formalize this problem and explore solutions that provide confidentiality after the full compromise of devices and user passwords. We propose two new solutions that use explicit key deletion and forward-secret protocols combined with key storage on porter devices. Our solutions provide the users with control over their privacy. We analyze the proposed solutions using an automatic verification tool. We also implement a prototype using a mobile phone as a porter device to illustrate how the solution can be realized on modern platforms.

Categories and Subject Descriptors

C.2 [**Computer Systems Organization**]: Computer-Communication Networks; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Unauthorized access*

General Terms

Design, Security

Keywords

Security Protocol, System Security, Full Compromise

1. INTRODUCTION

Confidential communication is a basic security requirement for modern communication systems. Solutions to this problem prevent an attacker that observes the communication between two parties from accessing the exchanged data. We address a related, but harder, problem in a scenario where the attacker is not only able to observe the communication between the parties, but can also fully compromise these parties at some time after the confidential data has been exchanged. If a protocol preserves confidentiality under such attacks, we say that it provides *forward secrecy under full compromise*. This is a stronger notion than forward secrecy [18], which guarantees confidentiality when participants' long-term secrets (but not their devices or passwords) are compromised. For example, a subpoena is issued and the communication parties must relinquish their devices and secrets after (e. g., e-mail) communication took place. In this scenario, the parties would like to guarantee that the authorities cannot access the exchanged information, even when given full access to devices, backups, user passwords, and keys, including all session keys stored on the devices.

Assuming public communication channels, any solution to the above problem must ensure that the communication is encrypted to prevent eavesdropping. The challenge in solving this problem is the appropriate management and deletion of the keys used to encrypt the data. Several solutions to this problem have been proposed. First, the Ephemerizer system [28] stores the encryption keys on a physically separate, trusted server accessible by all communicating parties. A drawback of this approach is that trust is placed in one entity, whose compromise would be disastrous for all parties using its services (e. g., companies and individuals). To address this concern, [21] proposes using Distributed Hash Table (DHT) networks for key storage and deletion, thereby removing trust from a central entity. This system, however, only provides probabilistic key deletion without guarantees on the deletion times of stored keys. Furthermore, researchers have shown how to attack this prototype implementation using Sybil attacks on DHTs, which enabled the attackers to reconstruct keys [36]. This attack highlights the problem of delegating key deletion to arbitrarily selected, untrusted nodes.

In this work, we formalize the problem of forward secrecy under full compromise and explore new solutions that provide confidentially after the compromise of devices and user passwords. Our solutions rely on the existence of trusted, reliable *porter devices* that manage encryption keys. We do not require that the principals trust one central server but enable the receivers to select their own key storage devices (based on their trust). We thus enable users to control their own privacy. Although it might seem that – given trusted porter devices – solutions to this problem would be simple, they turn out to be surprisingly complex. This complexity stems from (i) the need to ensure that the protocols do not allow key reconstruc-

ACSAC '10 Dec. 6-10, 2010, Austin, Texas USA

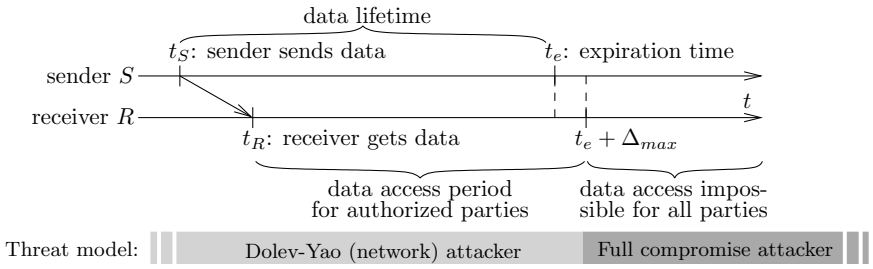


Figure 1: Timeline for time-limited data. Data can be accessed until its expiration time $t_e + \Delta_{\max}$, where t_e relates to the sender's clock and Δ_{max} is the maximal clock difference of the receiver with respect to the sender. After time $t_e + \Delta_{\max}$, data must be inaccessible to all parties, even under full system compromise.

tion under full compromise and (ii) the need to provide guarantees on the key deletion. Abstractly, our solutions use forward-secret subprotocols, session keys with different lifetimes, and timed, explicit key deletion as building blocks to achieve forward secrecy under full compromise. This prevents data access by all parties, including attackers, after a well-defined time. The requirement of guaranteed deletion motivates our use of porter devices: they enable timely key deletion even if the communication devices (e. g., PCs, laptops) cannot be guaranteed to be active.

Our main contributions are as follows. First, we formalize the concept of forward secrecy under full compromise. Second, we present two practical solutions to achieve it. Third, we formally analyze the presented solutions using an automatic verification tool [15]. Finally, we analyze their practical feasibility with a prototype implementation, using a mobile phone as porter device. We thus illustrate how the solution can be realized on modern platforms and how practical considerations can be handled.

The remainder of this paper is organized as follows. In Section 2, we specify the system requirements and our system and attacker models. In Section 3, we motivate our solution. We present our solution and formally analyze its properties in Section 4. In Section 5, we examine possible realizations and describe our prototype implementation. We discuss related work in Section 6 and draw conclusions in Section 7.

2. SYSTEM SPECIFICATION

2.1 Requirement Specification

Our goal is to design a system that provides data access only during a defined time period and afterward prevents access for all parties. We first introduce some key notions, which are illustrated in Figure 1.

Definition 1. The sender specifies data as *time-limited* by assigning a time after which the data must be inaccessible to the sender, the receiver, and any other party. We denote this time by t_e, also called the *expiration time*.

We note that t_e is relative to the sender's local clock.

Definition 2. During the lifetime of time-limited data, *authorized access* is granted only to parties that the sender selects as authorized to access the data.

Our system shall meet the following security requirement:

R1 **Time-dependent access control**: The time-limited data is *inaccessible* outside of the lifetime period specified by the sender.
   (a) During the data lifetime, only authorized access shall be granted.
   (b) After the data lifetime, no data access is possible for any party. This includes the sender, the receiver, and any compromised party.

We also define a functional requirement:

R2 **Data availability**: Given the successful communication between the sender and an authorized receiver (i. e., messages reach the intended recipient), the receiver can access the data during its lifetime.

2.2 System model

We consider the setting where a sender wants to transfer time-limited data to one or more receivers (the authorized recipients). The transfer may use any communication medium and include different applications, e. g., email exchange or server upload and download. A special case is the local storage of time-limited data as a form of self-communication involving only the sender. We make the following four assumptions:

Trusted communication partners. Communication partners, also called *principals*, follow the protocol. In particular, their devices timely and safely delete[1] data and they do not reveal time-limited data or keys in ways not specified by the protocol. Principals may shut down their communication devices and resume communication later, i. e., their devices need not be online at all times.

Authenticated communication. The sender and the receiver can communicate *authentically*. This may be achieved using pre-shared secret keys or authentic, pre-distributed (long-term) public keys. Pre-shared secrets are used to generate and verify message authentication codes (MACs) whereas long-term public keys are used for signature verification.

Trusted storage device. There exists a *reliable device* with an independent clock used for data (key) storage. Typical instances of such devices are built-in Trusted Platform Modules (TPMs), Hardware Security Modules (HSMs), or any external device, such as mobile phones, PDAs, or (e-banking) smartcards with readers (see also Section 5). Throughout this paper, we call this device a *porter* and denote it P. In our solution, the porter must be trusted in three ways: (i) P supports authentic communication, e. g., using authentic public keys or a physically secure channel, (ii) P supports the confidential storage and retrieval of data (in our protocols by the receiver), and (iii) P is regularly active and can provide autonomous, permanent erasure of stored data at specified times (or its inaccessibility after specified times). In general, the simpler the porter

[1]We assume that the principals use secure deletion [24,33] preventing data restoration.

device, the less complex its key deletion operation will be. At the same time, simple porter devices are, in general, more controllable and less error-prone than complex, general-purpose devices. We thus envision TPMs or dedicated smartcards as porters for corporate use and mobile phones or PDAs for (less critical) private use.

Loose time synchronization. The sender S, the receiver R, and the porter P are *loosely time-synchronized*. The local clock differences between the sender and the other principals at the data expiration time do not exceed Δ_{max}: when S's clock hits t_e, R's and P's clocks are between $t_e - \Delta_{max}$ and $t_e + \Delta_{\max}$. The principals' devices are not required to remain synchronized within Δ_{max} throughout the lifetime of the data but just at the expiration time. Time-limited data should be accessible at least until $t_e - \Delta_{max}$ and be inaccessible after $t_e + \Delta_{max}$.

2.3 Attacker model

We consider a two-phased attacker model. Our main aim is to model attackers capable of *full compromise* (introduced below), which models for example court orders or subpoenas. If such an attacker is present during the data access period, all protocols that require the data to be in the (accessible) device's memory are trivially insecure. We therefore design our protocols to provide security guarantees with respect to a two-phase attacker model (Figure 1): (i) *before* and *during* the data access period (defined by the sender), we consider a strong network (Dolev-Yao [19] type) attacker, and (ii) *after* the data access period, we consider an even stronger attacker capable of full compromise. Let $\mathcal{U}$ be the set of users authorized to access the transmitted data before its expiration time t_e.

Attacker model for $t \leq t_e + \Delta_{\max}$: *Active external attacker*. The attacker controls the network and may eavesdrop, intercept, inject, and block messages, but she has no control over the devices of users from the set $\mathcal{U}$. Users not in $\mathcal{U}$ may collude with the attacker and deviate from the protocol description. This attacker model corresponds to the standard Dolev-Yao model and is applicable to communication systems comprising ISPs, web mail providers, proxies, relay nodes, etc.

Attacker model for $t > t_e + \Delta_{\max}$: *Full Compromise*. In addition to controlling the network, the attacker completely controls the users' devices, including porter devices, and can compromise users' passwords and passphrases. The attacker can access and change all data stored on the devices and backups, possibly supported by court orders or subpoenas that oblige users to disclose data. In particular, she may compromise the principals' keys, including long-term and ephemeral secret keys, and she can coerce users to reveal the passwords used to secure decryption keys. We refer to this model as *full compromise*. This model is stronger than the Dolev-Yao model in that it allows the compromise of *all* data on the devices, including the data protected by user-selected passwords.

This two-phase attacker model is very strong. In many practical settings, the first-phase attacker will be weaker than a Dolev-Yao attacker. For example, it may be reasonable to assume that even in case of a subpoena after $t_e + \Delta_{\max}$, only communication logs were recorded in the phase before $t_e + \Delta_{\max}$ (e. g., by web mail or internet service providers), but no active attack was mounted. Indeed, such attacks often make evidence inadmissible. The concept of a phased attacker model also allows us to define other attackers that are stronger than the Dolev-Yao attacker in the first phase. In some scenarios the attacker might use a cryptographic attack to access principals' long-term secrets before getting full access to the devices at $t_e + \Delta_{\max}$. Although this is not part of our core attacker model, our solution even resists some attacks of this nature.

3. SOLUTION SPACE

In this section, we explore the space of possible systems that meet the requirements given in Section 2. We also introduce and categorize related work and motivate our solution.

Data transmitted over an open network cannot, in general, be explicitly deleted since the sender does not have access to (and may not even be aware of) all existing copies. Hence the sender must encrypt data before transmission to protect its confidentiality. Since an attacker (as defined in Section 2.3) may have full access to all devices after the data expiration time, data must also never be *stored* in plaintext on any device where it could possibly still reside after the time $t_e + \Delta_{\max}$. As principals can communicate authentically, they can use public-key cryptography to establish secret (session) keys over open networks and use the resulting keys to secure subsequent communication. The solution space therefore amounts to different ways of creating, managing, and deleting decryption keys.

Intuitive Approaches. We first look at two approaches for key management and deletion that appear intuitive but are inappropriate as solutions.

(1) The sender and receiver delete the established key immediately after the encryption and decryption phases, respectively.

This approach does not fulfill requirement R1.b (Section 2) if the encrypted data sent by the sender S arrives at receiver R after $t_e + \Delta_{\max}$ or if it never arrives at R (e. g., due to message blocking or delay attacks, transmission failures, or R being offline / inactive). In this case, the pre-agreed key K remains stored on R because the receiver never starts the decryption phase. This reveals the time-limited data under full compromise after $t_e + \Delta_{\max}$.

(2) The sender and receiver delete the key at its lifetime expiration t_e, e. g., using a job or task scheduler such as Cron.

This does not guarantee requirement R1.b because these automated tasks are not guaranteed to succeed. For example, users' personal computers usually have periods of inactivity during which they are turned off or they may have to be handed in for repair. In such cases, R may be turned off at the expiration time and system processes cannot erase expired keys from the device memory and disks.

From the above considerations, we conclude that the key K used to encrypt the time-limited data cannot be stored on either S or R. Hence it must be stored externally.

Related Work. We briefly review selected related work to illustrate relevant parts of the solution space. In the Ephemerizer system [28] and its application to file deletion [29], a physically separate, trusted machine, the *Ephemerizer*, generates and stores the keys used to encrypt and decrypt the data. Users interact with the Ephemerizer in order to retrieve the encryption or decryption keys. A potentially large number of users, for example a company's employees, use the same (logical) key generator and storage.

The authors of Vanish [21] propose using a de-centralized key storage based on peer-to-peer networks and DHTs. In their system, the sender picks a random encryption key, splits it using secret sharing, and stores the key shares in a DHT network from where the receiver can retrieve them as long as they exist. Due to the natural churn in such networks, the keys are eventually deleted.

Solution Dimensions. We identify four properties of key storage devices: (i) storage type, (ii) access options, (iii) level of guarantees for key management, and (iv) scalability. In the remainder of this section we explain these properties and show in Table 1 how they apply to the approaches above and to our solution.

(i) Storage type. The storage may be *centralized* (e. g., a remote server [28]), or *distributed* [21]; distributed storage requires key sharing. While deletion on a centralized storage is a well-

	Ephemerizer [28]	Vanish [21]	Our solution (Section 4)
Storage type	centralized/shared	distributed/shared	personal
Key generation	by the storage server	by S	by S and R or by R
Key deletion	deterministic	probabilistic	deterministic
Access to K	both S and R	both S and R	R (or S)
	over an open network	over an open network	over open/trusted networks
Scalability	scales (special-purpose)	limited (secondary with many users)	scales (special-purpose or secondary with few users per storage)

Table 1: Dimensions of the key storage and their instantiation by different solutions. Our solution allows access to the encryption key K by R but can easily be extended to enable access also by S on a separate storage (belonging to S).

defined operation, providing guarantees on the deletion of (sufficiently many) key shares on distributed storage is challenging.

(*ii*) Access. The storage may be *personal* or *shared*. Personal storage allows exclusive access by either S or R. The access to personal storage may be based on *public* or *secure* channels; an example for the latter are independent storage units within a user's device. Shared storage (e.g., a network server) permits multiple parties to store and retrieve data. We do not consider storage that only S and R can access because it is a special case that could be used to directly transfer time-limited data. The communication channels to access shared storage are typically public. Since the key must be stored in plaintext on shared storage[2], it may allow attackers to collect data before the expiration time and use it later to access the data. The attack [36] on Vanish is an example of this.

(*iii*) Guarantees on key management. Any key storage must store and manage keys and delete them in a timely way. We distinguish between *deterministic* and *probabilistic* key deletion. In contrast to probabilistic key deletion, deterministic key deletion provides guarantees on the times when keys will be deleted; it is typically harder to achieve on complex or distributed systems (e. g., network servers) than on simple, monolithic devices.

(*iv*) Scalability. The storage should provide functionality for a large number of users without substantially degraded performance. We distinguish *special-purpose* storage that can be designed to scale well with the number of users (e. g., [28]) and *secondary storage* that fulfills different primary purposes and, additionally, provides key management. In the latter case, the primary functions may degrade with the additional key management functionality of the storage; in this case, the scalability is limited.

4. OUR SOLUTION

4.1 Solution Overview

As motivated in Section 3, the sender encrypts the time-limited data prior to transmission. The encryption key is established on a per-message basis between S and R using an authenticated Diffie-Hellman (DH) key establishment protocol. In our solution, we relocate the encryption keys to an autonomous porter device under the receiver's control (we do not use a central server because it requires the users' trust and creates a single point of failure). The porter device will independently delete keys once the expiration time of the messages encrypted using those keys is reached. Given that the porter possesses the sole copy of this encryption key at the expiration time and the porter will delete keys when they expire, this approach prevents data access by any party after $t_e + \Delta_{\max}$.

[2]If the decryption key K was encrypted, this would bring us back to the original problem: how and where to store the key. Asymmetric encryption with R's long-term public key would not resist a full-compromise attack after $t_e + \Delta_{\max}$.

A porter-based approach requires elaboration to provide authenticity and forward-secrecy for the connections from the sender to the receiver and between the receiver and the porter. This requires carefully managing multiple short-term keys. In security applications, e. g., off-the-record messaging [3, 11], short-term keys are created on demand and deleted immediately after the data encryption and decryption. Deleting the decryption key after the data transmission is, however, not a solution in our scenario: we must ensure data inaccessibility after the expiration time $t_e + \Delta_{\max}$ even if the sender's message is not received before $t_e + \Delta_{\max}$ (see Section 3).

4.2 Forward Secrecy under Full Compromise

We introduce the notion of forward *forward secrecy under full compromise* and explain why we need it. *Forward secrecy* means that the compromise of the principals' long-term private keys does not compromise past session keys [18, 27]. Our system requires forward secrecy not only under the compromise of long-term keys but also under *full compromise* (as defined in Section 2.3) after the expiration time. Given this extended notion of compromise, we similarly extend the definition of forward secrecy.

Definition 3. A protocol is *forward-secret under full compromise* with respect to time-limited data m if the full compromise of the involved principals and their devices after the data expiration time does not compromise the secrecy of m.

Forward secrecy under full compromise is a stronger property than (standard) forward secrecy because it also accounts for the principals' internal states after the expiration time. As a consequence, time-critical data and the respective encryption keys must be erased from the principal's devices such that they are nonexistent at the expiration time. *Key* and *data deletion* must be part of any protocol that provides forward secrecy under full compromise. A second essential component concerns those parts of the protocol that involve session keys, which we call *subprotocols*, e. g., for key establishment. Forward secrecy under full compromise requires that all subprotocols used to establish session keys for data encryption provide *forward secrecy*.

4.3 Protocol

We now present the main idea of our protocols. We focus on the case where the receiver uses the key storage (rather than the sender). Figure 2 provides a protocol sketch that we will later instantiate with concrete solutions. All delete commands are secure deletions. We consider the following four protocol phases:

1. **Key establishment**: The sender S defines the data lifetime t_e and agrees with the receiver R on the mid-term key K (or a key pair where K is the decryption key). R initiates the safe storage of K along with t_e on the porter P and deletes

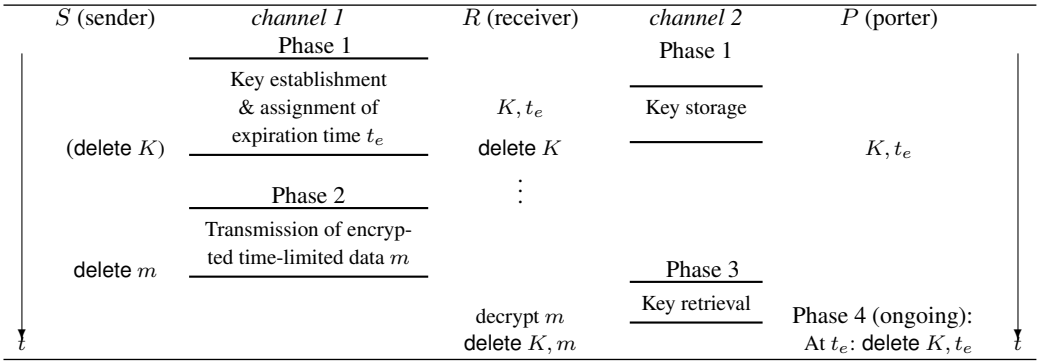


Figure 2: Protocol sketch. The basic building blocks are commands for explicit, secure deletion and forward-secret subprotocols during the communication phases (Phases 1–3).

its own copy of K.[3] If the key establishment involves key contributions from the principals, the ephemeral private keys are deleted right after the key establishment.

2. **Communication/storage**: S transmits the data m, encrypted using key K, and then deletes both the plaintext and K.
3. **Data access**: Upon receiving the encrypted data, R attempts to retrieve K from P in order to decrypt m. After successful data access, R deletes both the plaintext m and K. This phase may occur multiple times.
4. **Key management/deletion**: In parallel with phases 1–3, P permanently deletes keys from its storage once they expire.

Our solution involves three kinds of keys for different time intervals:

1. a *mid-term* encryption key K (or key pair) for encrypting and decrypting time-limited data,
2. *long-term* authentication keys used to authenticate the messages, and
3. *short-term* (ephemeral) session keys to provide secrecy of the communication between the principals and to the porter.

The notion of a *mid-term* key is non-standard but is appropriate for our key K, which must exist during the data's lifetime and is permanently erased thereafter.

Encryption using mid-term keys can be based on symmetric or public-key cryptography. We will provide examples of both in Section 4.4. The examples also differ in the assumptions on the communication channels underlying the protocols. We require two channels: one between the sender and the receiver for data transmission and key-establishment and a second channel between the receiver and the porter for key storage and retrieval. We introduce two common channel types in the following; based on the available channels, different subprotocols will provide forward-secrecy.

Physically secure channel: A physically secure (PS) channel provides confidentiality and authenticity without cryptographic measures. An example of such a channel is a shielded wire that connects the receiver's motherboard to a trusted hardware module. Due to the physical security of the communication, forward secrecy is trivially achieved because no long-term or short-term keys are involved in the communication.

Dolev-Yao channel: A Dolev-Yao (DY) channel is subject to attacks under the Dolev-Yao attacker model, involving eavesdropping, message corruption, insertion, and blocking (erasing). An example of a DY channel is a wireless (e. g., Bluetooth) connection between two devices.

[3]In our protocols, R stores and retrieves the key. In a different protocol, S may also store the key on a porter of its own.

The standard way to achieve forward secrecy on a Dolev-Yao channel is to establish ephemeral encryption keys, typically by using an authenticated DH protocol [12], and to discard them after their use. In this case, the ephemeral DH public keys g^{r_S} and g^{r_R} are exchanged and stored only during the key establishment. They are destroyed thereafter along with the private keys r_S and r_R. The established key $K = g^{r_S r_R} = g^{r_R r_S}$ is the encryption key.

4.4 Protocol Instances

We now present two instances of the protocol sketch of Figure 2, shown in Figures 3 and 4. The two protocols differ in how they achieve forward secrecy on the communication channels between S, R, and P.

We use the following notation: $[M]_K$ and $[M]_K^{-1}$ denote the symmetric encryption and decryption of a message M with key K. $\mathcal{A}_S(M)$ denotes that message M is authenticated by principal S (described below). Communication is expressed as $S \xrightarrow{M} R$, meaning that S transmits message M to receiver R. The tupling of multiple data items in a message is denoted by ",". For DH key establishment, g denotes the public generator of the group used, r_S is the ephemeral private key of principal S, and g^{r_S} is S's ephemeral public key; the use of the modulus (mod n) is implicit.

4.4.1 Protocol 1

Protocol 1 (Figure 3) is designed to be used when S, R, and P communicate over DY channels. In this scenario, P may, e. g., be a mobile phone that belongs to R. Protocol 1 uses symmetric encryption to transmit the time-limited data m. The protocol is initiated by S, who starts a DH key establishment with R. R then establishes another ephemeral DH key L with the porter device P and uses it to send K encrypted to P. Later, after receiving the encrypted time-limited data from S, R establishes a new ephemeral key L' with P and uses L' to retrieve K. For each subsequent retrieval of the encryption key K, a new ephemeral key is established.

The DH key exchanges of Protocol 1 follow the standard two-way ISO-9798-3 protocol [23].[4] We do not require a third message for key confirmation in which the sender returns both ephemeral public keys to the receiver to confirm that it possesses the same key. Under our attacker model, the receiver is not compromised before it sends its DH key contribution (when t_e expires, both parties abort the protocol).

The following components are essential to Protocol 1:

[4]The standard also specifies a random index i into a universal hash function family H in message 2, so that the shared key computed is $K = H_i(g^{r_S r_R})$. We do not use this.

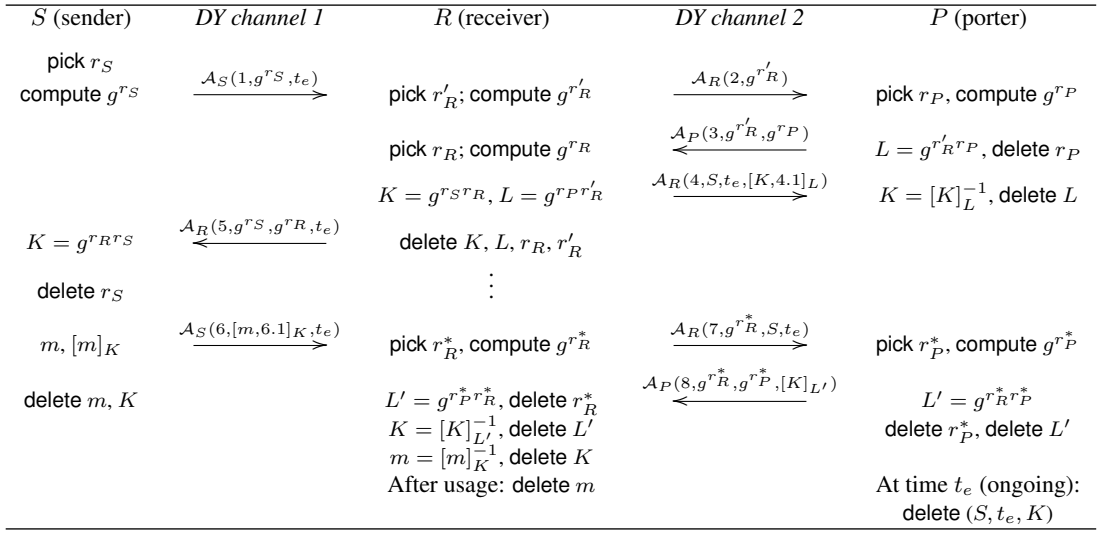


Figure 3: Protocol 1. The protocol can be run over two Dolev-Yao (DY) channels, between S and R and between R and P. The established symmetric mid-term key K is used by S to encrypt the time-limited data m. All messages are authenticated, denoted by the authentication function $\mathcal{A}_X(\cdot)$, which represents the function input concatenated with a digital signature of principal X.

1. All messages are **authenticated** by the transmitter, as indicated by the authentication function $\mathcal{A}_X(\cdot)$ where $X \in \{S, R, P\}$ is the authenticating principal. This can be achieved using message authentication codes (MACs) with pre-shared symmetric keys or by digital signatures using X's long-term secret key. In the latter case, the first message would be $1, g^{r_S}, t_e, Sig_S(1, g^{r_S}, t_e)$, where $Sig_S(M)$ denotes the digital signature of M using S's long-term key.
2. The principals **verify** the authenticity of received messages (by verifying signatures or MACs) and check the validity of t_e. The principals abort the protocol if t_e has expired or if message authenticity cannot be verified.
3. If the protocol aborts due to failed time or authenticity checks, **abortive measures** must be taken. In particular, critical data (such as encryption keys and DH key contributions), which may be present on a device, must be securely deleted.

Encrypted vs. plain storage of K. In Protocol 1, the mid-term encryption (decryption) key is stored in plaintext on P and revealed only to R (encrypted over the DY-channel). While the unencrypted storage of K may seem like a weakness, under our attacker model, full device compromise only occurs after the expiration time when K is already deleted. We still consider it realistic that the porter device (e. g., a mobile phone) may be lost or stolen before the expiration time; in either case, we can assume that the owner is aware of the loss. To preserve data privacy in this case, we propose to store K *encrypted* on P: In Protocol 1, the receiver would send the encrypted key K to the porter (i. e., $[[K]_X]_L$ instead of $[K]_L$) and store the symmetric encryption key X along with the expiration time t_e of K on R. Whenever the owner notices the loss of his porter device, he can delete X from R's disk.

4.4.2 Protocol 2

Protocol 2 (Figure 4) assumes a physically secure (PS) channel between R and P; e.g., P could be a TPM directly connected to R's computer. Thus no key agreement is required on this channel. Furthermore, Protocol 2 uses *asymmetric* encryption to secure time-limited data (independent of the PS channel).

In Figure 4 we use the notation from the beginning of this section. Additionally $\{M\}_{PK_R^+}$ (and $\{M\}_{PK_R^-}$) denote the public-key encryption (and decryption) of message M with the public (private) key PK_R^+ (PK_R^-) of principal R, respectively. Protocol 2 is based on R's authenticated broadcast, indicated by *, of the mid-term public keys PK_R^+. These public keys form part of freshly generated key pairs and are broadcasted along with their expiration times t_e. An example broadcast is the authenticated publication of PK_R^+ along with t_e on the receiver's website, or the receiver's reply to a request by the sender (not shown in Figure 4). The corresponding secret keys PK_R^- are not stored on R but on a porter directly connected to R over a PS channel. At any point in time, the sender may pick the public key that corresponds to the desired data lifetime, use it to encrypt the time-limited data, and transmit the message to the receiver, along with the respective expiration time (thereby enabling the receiver to identify the right secret key). The messages transmitted over the DY channel are authenticated.

4.4.3 Comparison

Protocols 1 and 2 differ in (i) how they create the mid-term encryption key and (ii) how they achieve forward-secrecy on the communication channel between R and P.

Protocol 1 uses key contributions by both the sender and the receiver to establish the symmetric encryption key and assumes a DY channel between R and P. This requires DH session key establishments on both communication channels. A typical application for Protocol 1 is the forward-secure email-communication of a company (under our full-compromise attacker model) using a trusted remote device for key management, e. g., a key card or other special-purpose devices.

In contrast to this, Protocol 2 uses asymmetric encryption with key pairs created by the receiver. The public keys may, e. g., be announced on a private user's webpage. Protocol 2 does not require DH key establishment on the communication channel between S and R. Due to the PS channel, it also does not require DH key establishment between R and P. The communication devices using Protocol 2 must be able to perform public-key operations; for example, the porter can be a TPM attached to R. In a slightly dif-

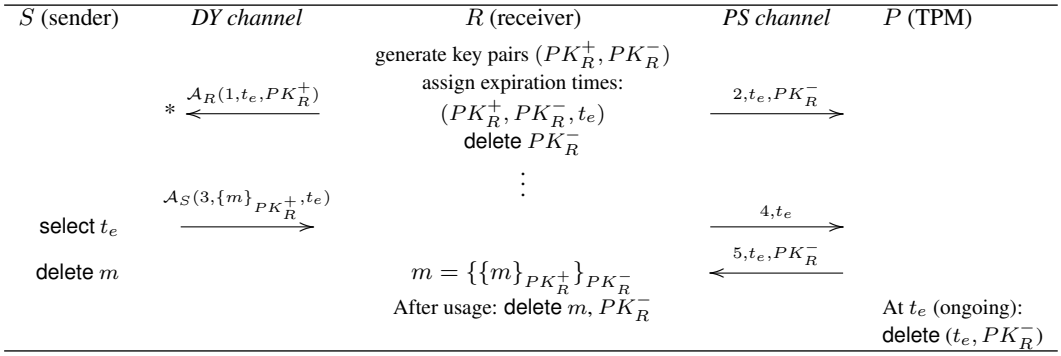


Figure 4: Protocol 2. The protocol assumes a physically secure (PS) channel between R and P (e. g., P is an HSM physically wired to R's hard disk). Hence, messages between S and R need not to be further protected by encryption or authentication measures. The mid-term key pair used for data encryption and decryption is (PK_R^+, PK_R^-). There are no ephemeral keys used.

ferent setting, this protocol can also be applied if S and R are porter devices that can directly communicate. In this case, the operations on the PS channel are simple storage and data retrieval operations to and from the memory of the porter.

In summary, if the communication devices can perform key management, they can also be used for key storage; if not, the key management should be outsourced to a suitable porter. We also note that the building blocks of Protocols 1 and 2 can be mixed, e. g., one can build an implementation that uses symmetric encryption while relying on a TPM connected by a PS channel.

4.5 Formal Protocol Analysis

We now construct formal models of our protocols and analyze the secrecy of the message m with respect to our attacker model (Section 2.3) using the Scyther tool [15]. We chose Scyther since it provides support for revealing the principals' states and enables us to analyze forward secrecy under full compromise [6, 7]. We first provide background information on Scyther.

4.5.1 Background on Scyther

Scyther is a tool for the symbolic automatic analysis of the security properties of cryptographic protocols (typically confidentiality or variants of authenticity). It assumes perfect cryptography, meaning that an attacker gains no information from an encrypted message unless she knows the decryption key (this is a standard assumption in symbolic methods). Scyther takes as input a role-based description of a protocol in which the intended security properties are specified using *claims*. Claims are of the form claim(Principal,Claim,Parameter), where Principal is the user's name, Claim is a security property (such as 'secret'), and Parameter is the term for which the security property is checked.

Recent versions of Scyther can analyze protocols with respect to a family of attacker models, ranging from a standard Dolev-Yao style network attacker to stronger attackers capable of various types of compromise. The attacker model is specified by selecting a set of attacker capabilities, such as revealing the short-term or long-term secrets of users. To analyze our protocols, we enable the following attacker capabilities: (i) Long-term key reveal for all principals and for other parties *after* the protocol execution, (ii) Session (short-term) key reveal for all parties not part of the current protocol execution, and (iii) Session-state reveal, which reveals the entire contents of the session-state of the parties. Together, these capabilities model the attacker from Section 2.3.

For most protocols and properties, the tool either finds an attack or establishes the unbounded verification of the protocol's properties with respect to the specified attacker model. In the remaining cases, bounded verification is performed where the *bound* defines the number of considered runs, i. e., the maximum number of parallel threads (or executions of role descriptions) executed by honest principals. This bounded result is similar to model-checking approaches for formal protocol verification. Attacks such as replay or man-in-the-middle attacks are typically found within the bound of two or three runs for many protocols (e. g., [5])[5]. The verification of over 100 protocols in [16] showed that no attacks were found that involved more runs than the number of principals in the protocol (except for protocols specifically constructed as counterexamples).

4.5.2 Analysis of Protocol 1

We model Protocol 1 (Figure 3) using eight send and receive events for the three principals S, R, and P. The complete protocol models and the tool itself are available at [2]. To give some intuition, we display the part that models the sender S:

```
role S {
  const rS: Nonce;           // S's DH key contribution
  const te: Nonce;           // expiration time
  const M:  Nonce;           // time-limited data
  var beta: Ticket;          // R's DH key contribution

  claim_sidS(S, SID, te);    // mark T_e as session id
  // Phase 1
  send_1(S,R,g1(rS),te,{l1,g1(rS),te}sk(S));
  recv_5(R,S,g1(rS),beta,te,{l5,g1(rS),beta,te}sk(R));
  // Phase 2
  send_!6(S,R,{l6a,M}g2(beta,rS),te,
    {l6b,{l6a,M}g2(beta,rS),te}sk({S});
  claim_s(S, Secret, M);
}
```

When using Scyther, security properties are modeled as local properties: If an agent executes a particular role, what can be concluded about the state of other agents or the attacker's knowledge? Here we analyze whether the protocol ensures the secrecy of m after the execution of an instance of S or R, and the secrecy of K after the execution of an instance of P, both under full compromise. In particular, we verified the following claims: S: claim(S,Secret,m), R: claim(R,Secret,m), and R: claim(P,Secret,K). As Scyther currently does not support explicit key expiration times, we model the expiration as happening immediately after the protocol execution, i. e.,

[5]The security analysis in [5] indicates that the Ephemerizer protocol is secure in terms of secrecy but insecure regarding integrity. The analysis is based on two (or three) runs.

Figure 5: Scyther result for Protocol 1.

after the key is retrieved from P. This is a worst case model because early key expiration only gives the attacker more knowledge at earlier times and thus more possibilities for attacks.

Figure 5 shows the results of the Scyther analysis. Scyther validates that no attacks exist on the model of Protocol 1 that involve less than four honest agent runs. For bounds of four or more parallel runs, the verification process did not terminate (within a day) due to the complexity of the analysis. Similar to bounded model-checking this neither falsifies the protocol nor proves its correctness, but establishes that no attacks exist within the given bounds.

4.5.3 Analysis of Protocol 2

We model Protocol 2 (Figure 4) in Scyther by two send events over the DY channel. Messages over the physically secure channel are not modeled as events because they are not subject to compromise (as opposed to the DY channel). Consequently, we verified the following two claims: S: claim(S,Secret,m) and R: claim(R,Secret, m). The input file provided to the Scyther tool can be found at [2]. The automatic analysis validates that no attacks exist on the model of Protocol 2 that involve less than ten honest agent runs. For bounds of ten or more parallel runs, the verification process did not terminate (within a day).

5. IMPLEMENTATION AND EVALUATION

We now examine possible realizations of key storage devices (Section 5.1), describe results from our mobile phone prototype implementation (5.2), and evaluate our solution (5.3).

5.1 Possible Realizations of Porter Devices

Dedicated Platforms. One possible realization of the porter device uses a platform that is embedded in the receiver device or is (occasionally) attached to the receiver. Example platforms on which porter functionality (i. e., key storage and delayed deletion) can be implemented are dedicated hardware security modules ([1] is an example of a platform that offer porter functionality). Note that our solution does not assume that the porter is tamper-resistant. Existing TPM modules could be used as porters but their functionality would have to be extended with delayed key deletion. TPMs used for this purpose must have an internal battery and clock; these are typically available in more advanced platforms (e. g., the IBM 4758 Cryptographic Coprocessor [14]). Typically, the communication with such a dedicated platform is a physically secure channel (a wired link); thus no additional measures are needed to guarantee the forward secrecy of the communication to and from the porter.

Mobile Phones. Private users might not have access to dedicated platforms. However access to mobile phones is widespread, so they are natural candidates for porter devices. For most users, their primary mobile phone is always operational and users pay close attention to their correct functioning and charging. The usability of mobile phones and personal computers has improved over the years and there are many convenient (wireless and wired) channels through which these devices can communicate and synchronize (e.g., Bluetooth [10]). The storage requirements of our solution are easily met with today's smartphone platforms (see Section 5.2). The communication between a mobile phone porter and the user's device must be forward-secret. Message secrecy is preserved, even given mobile phone loss prior to key deletion, as discussed in Section 4.3.

5.2 Prototype Implementation

To demonstrate the practical feasibility of our solution, we developed a prototype implementation of Protocol 1 for the communication between the receiver and the porter device. Our porter is a NexusOne [35] mobile phone (firmware 2.1, kernel 2.6.29, Android OS, 512 MB memory), depicted in Figure 6a. The receiver is implemented on a laptop running MacOSX 10.6.2.

The communication between the receiver and the porter is based on Bluetooth, using the Bluecove library [9], which we recompiled for a 64-bit Mac. Cryptographic operations are implemented using the Bouncycastle [4] library. To secure the key K on the wireless channel, we use symmetric AES/CBC/PKCS5 encryption with a 256-bit key that is derived from the established ephemeral DH key L (or L') by a SHA-256 hash. All messages are authenticated by a 32-byte HMAC-SHA with a 224-bit key that is pre-shared between the laptop and the mobile phone. Furthermore, we use 2-byte packet IDs, 8-byte timestamps, a 16-byte initialization vector for AES, and 1024-bit DH-key pairs. We use base64 encoding to transmit the packets over an RFComm Bluetooth connection and the tool `srm` to perform secure deletions on the laptop.

Figure 6b displays the execution times and the standard deviation for 100 runs as measured on the laptop. We see that key storage and retrieval are below one second, once the Bluetooth connection has been established. The times for key storage have a larger variance than for key retrieval since files are created and written rather than just read.

The key storage and delayed deletion functionality on the porter is implemented as follows. We store the keys along with their expiration times in files. To ensure the timely deletion of a file (i. e., key), we set an alarm service to automatically trigger the deletion of the respective file upon the timestamp's expiration. If the phone is shut down and rebooted before the expiration time, a background process (triggered by the boot-complete system broadcast) parses the key files, deletes files with expired timestamps, and resets the alarms. Figure 6c displays the time for setting the system alarm for different numbers of keys (files). We see the linear dependency of the number of keys on the alarm reset time. This background process does not significantly degrade the usability of the device.

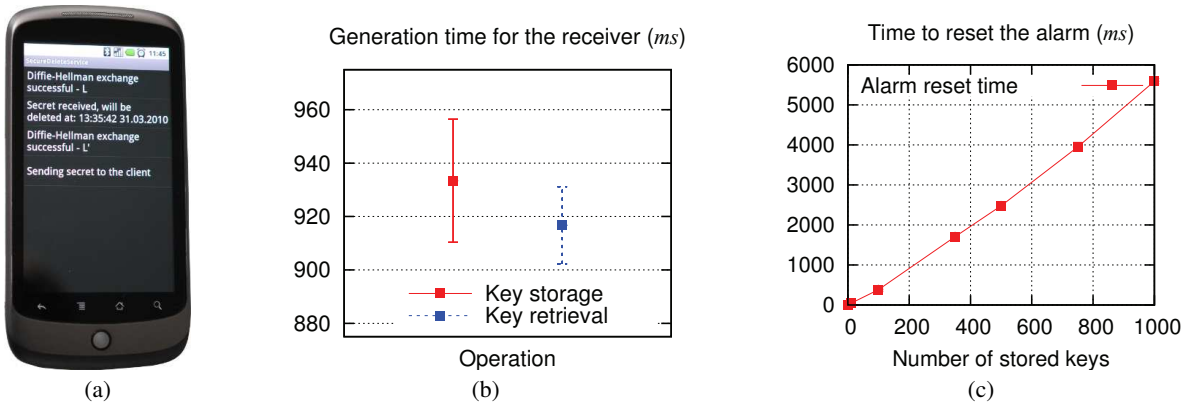
Given the execution times in Figure 6 and a consumption of 1.5 kB for program storage and 0.18 kB per key, our prototype implementation confirms the usability of our approach in practice.

Note on Secure Deletion

When exploring implementation options, we observed that many embedded devices have limited functionality for secure deletion due to OS characteristics (like versioning) or hardware specifics (e.g., NAND storage often uses log-structured sequential writes). Enabling secure deletion on these devices is subject to recent research, examples include [26] for the Android YAFFS file system, [30] for versioning file systems, and [34] regarding data remanence in flash memory devices. Secure deletion may not be realized on certain off-the-shelf devices and care should thus be taken in the selection of the porter device.

5.3 Integration with Applications

Given a functional porter device, our solution can be integrated



(a) (b) (c)

Figure 6: (a) Prototype implementation of the key storage (porter) functionality on a NexusOne mobile phone. (b) Protocol execution times for the receiver. The plot shows the average times and the standard deviation for 100 runs of Protocol 1. (c) Time for resetting the system alarm (used to delete keys at their expiration time) after a phone reboot for different numbers of keys.

in many applications, such as file storage, web services, and e-mail.

File Storage. The simplest application of our solutions is to local file storage, where a device locally stores confidential data that should be inaccessible after the expiration time. To enable this, the device encrypts the data locally with a key stored on a porter device. Here, our protocols can be substantially simplified: the only communication that needs to be forward-secret is that between the device and the porter (this can be further simplified if this channel is physically secure). Remote file storage is similar to local file storage in that the only device that has access to the data decryption keys is the device that created them. However, the communication between the user's device and the remote file server where the data is stored must be forward-secret.

Web Services. Another application is where users share their data (e. g., pictures, movies, files) using remote storage in the form of a web service. The data is to be kept secret even in the case of full device (sender and receiver) and service compromise. In contrast to standard remote file storage, here the communication key must be agreed upon between the sender (who uploads the data) and the receiver(s) (who download the data). The receiver stores the key on a porter device and obtains it when it downloads the data. The communication used for key agreement must be forward secret.

E-mail. Finally, we consider the scenario where the sender and the receiver wish to preserve the secrecy of their e-mail correspondence. In this scenario, the parties first agree on the keys that they will use for their communication and on their expiration times. They then store the keys on their respective porter devices and exchange e-mail. Although we could directly use Protocol 1, it can be optimized by making a mobile phone porter establish the keys directly with the sender. This can be alternatively done via e-mail exchange, without the participation of the receiver. The receiver could be notified that the keys are established when it synchronizes (e. g., over an IMAP server) with the e-mail communication.

6. RELATED WORK

Shoup [32] defined three notions for the compromise of principals: *static corruptions* (in which principals are either compromised or not), *adaptive corruptions* (in which long-term keys may be compromised), and *strong adaptive corruptions* (in which the compromise of principals reveals both long-term and short-term stored secrets). In our attacker model, we build on the third notion by considering full device and user password compromise after a specific time.

Methods for protecting data confidentiality under device compromise include secret sharing [31], threshold cryptography [17], and forward secrecy [22]; we focus on the last method. Canetti et al. [13] proposed a forward-secure public-key encryption scheme in which a receiver evolves its private key such that it can only decrypt messages with a later timestamp. A similar idea was adopted by Bellare et al. [8] for forward-secure digital signature schemes. We cannot use such approaches because private keys cannot evolve when devices are inactive.

The confidentiality of data exchanged between individuals or organizations is attracting increasing attention. Centralized systems such as [20] for server-based sharing and storage of personal data offer access control and data deletion at user-defined or automatically derived times. However, they require full trust in the service provider to treat passwords and data confidentially and to delete both when specified. Ephemerizer-based solutions [25, 28, 29] similarly require trust in a central server. As discussed in [21], this does not ensure the data confidentiality in the presence of service-provider mismanagement and legal action to reveal data.

7. CONCLUSION

We addressed the problem of data confidentiality in scenarios where attackers can observe the communication between principals and can also fully compromise the principals after the data has been exchanged, thereby revealing the entire state of the principals' devices. We explored the design space of solutions to this problem and proposed two protocols that use key storage on porter devices along with explicit deletion and forward secret subprotocols to achieve secrecy under full device, user and communication compromise. The solutions provide users with full control over their data privacy. We formalized our proposed solutions and analyzed them using an automatic verification tool. Our prototype implementation shows their practicality and feasibility.

Acknowledgment

The authors thank Claudio Marforio for his work on the implementation of the prototype.

8. REFERENCES

[1] Privat Server HSM (Hardware Security Module). http://www.arx.com/products/hsm.php.

[2] Scyther protocol models for keeping data secret under full compromise using porter devices. http://people.inf.ethz.ch/cremersc/scyther/DataDeletion, Oct 2010.

[3] Chris Alexander and Ian Goldberg. Improved user authentication in off-the-record messaging. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 41–47, New York, 2007. ACM.

[4] Bouncy Castle Crypto APIs. http://www.bouncycastle.org.

[5] Charu Arora and Mathieu Turuani. Validating integrity for the Ephemerizer's protocol with CL-Atse. In *Formal to Practical Security: Papers Issued from the 2005-2008 French-Japanese Collaboration*, pages 21–32. Berlin, 2009.

[6] David Basin and Cas Cremers. Degrees of security: Protocol guarantees in the face of compromising adversaries. In *Proceedings of the 24th International Workshop on Computer Science Logic (CSL)*, pages 1–18. Springer, 2010.

[7] David Basin and Cas Cremers. Modeling and analyzing security in the presence of compromising adversaries. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, pages 340–356. Springer, 2010.

[8] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO)*, pages 431–448, London, 1999.

[9] Bluecove. Java library for Bluetooth (JSR-82 implementation). http://bluecove.org.

[10] Bluetooth SIG, Inc. Bluetooth specification version 3.0 + HS, 2009.

[11] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use PGP. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 77–84, New York, 2004. ACM.

[12] Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 2003.

[13] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 255–271. Springer, 2003.

[14] IBM Corporation. IBM PCI Cryptographic Coprocessor. General Information Manual. http://www-03.ibm.com/security/cryptocards.

[15] Cas Cremers. Scyther. A tool for the automatic verification of security protocols. http://people.inf.ethz.ch/cremersc/scyther.

[16] Cas Cremers. *Scyther—Semantics and Verification of Security Protocols*. PhD thesis, Eindhoven University of Technology, 2006.

[17] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *Proceedings on Advances in Cryptology (CRYPTO)*, pages 307–315, New York, 1989. Springer.

[18] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.

[19] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[20] Drop. Simple real-time sharing, collaboration, presentation. http://drop.io.

[21] Roxana Geambasu, Tadayoshi Kohno, Amit Levy, and Henry M. Levy. Vanish: Increasing data privacy with self-destructing data. In *Proceedings of the 18th USENIX Security Symposium*, pages 299–315. USENIX Association, 2009.

[22] Christoph G. Günther. An identity-based key-exchange protocol. In *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology (EUROCRYPT)*, pages 29–37, New York, 1990. Springer.

[23] Prateek Gupta and Vitaly Shmatikov. Key confirmation and adaptive corruptions in the protocol security logic. In *Proceedings of the Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA)*, 2006.

[24] Peter Gutmann. Secure deletion of data from magnetic and solid-state memory. In *Proceedings of the 6th USENIX Security Symposium (SSYM), Focusing on Applications of Cryptography*, pages 77–90, Berkeley, California, 1996. USENIX Association.

[25] Disappearing Inc. http://www.disappearing-inc.com.

[26] Jaeheung Lee, Junyoung Heo, Yookun Cho, Jiman Hong, and Sung Y. Shin. Secure deletion for NAND flash file system. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 1710–1714, 2008.

[27] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[28] Radia Perlman. The Ephemerizer: Making data disappear. *Journal of Information System Security*, 1:51–68, 2005.

[29] Radia Perlman. File system design with assured delete. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. ISOC, 2007.

[30] Zachary N. J. Peterson, Randal Burns, Joe Herring, Adam Stubblefield, and Aviel D. Rubin. Secure deletion for a versioning file system. In *Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST)*, pages 143–154, Berkeley, California, 2005. USENIX Association.

[31] Adi Shamir. How to share a secret. *Commununications of the ACM*, 22(11):612–613, 1979.

[32] Victor Shoup. On formal models for secure key exchange. Research Report RZ 3120, IBM Research, 1999.

[33] Muthian Sivathanu, Lakshmi N. Bairavasundaram, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Life or death at block-level. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 379–394, Berkeley, California, 2004. USENIX Association.

[34] Sergei Skorobogatov. Data remanence in flash memory devices. In *Proceedings of the Cryptographic Hardware and Embedded Systems Workshop (CHES)*, pages 339–353, 2005.

[35] NexusOne Smartphone. http://www.htc.com.

[36] Scott Wolchok, Owen S. Hofmann, Nadia Heninger, Edward W. Felten, J. Alex Halderman, Christopher J. Rossbach, Brent Waters, and Emmett Witchel. Defeating Vanish with low-cost Sybil attacks against large DHTs. In *Proceedings of the 17th Network and Distributed System Security Symposium (NDSS)*. ISOC, 2010.