COMPUTING SCIENCE

How to Delete a Secret

Feng Hao and Dylan Clarke

TECHNICAL REPORT SERIES

No. CS-TR-1364    December 2012
How to Delete a Secret

F. Hao and D. Clarke

Abstract

This paper discusses the secure data erasure problem and introduces a new Proof of Deletion protocol, which assures secure data deletion in a way that any independent third party can verify cryptographically. The use of cryptography in data deletion is not new, but previous solutions are unsatisfactory as they merely return a single bit: whether the deletion is successful. However, this single bit is not easily verifiable. The implementation of the underlying deletion mechanism is entirely opaque to a user. This is particularly the case when the encryption program is encapsulated within a tamper resistant chip. Furthermore, all existing solutions claim only to make "best efforts" to delete data, but without any commitment to the outcome. Our protocol systematically addresses all these issues. First, we provide an auditing function to facilitate a user verifying that the encryption was done correctly. Second, instead of returning just one bit like all previous data deletion schemes, our solution returns a proof of deletion that is universally verifiable. This returned proof formalizes the commitment of the storage system in erasing data and cryptographically binds this commitment to the outcome of the operation.
Abstract

This paper discusses the secure data erasure problem and introduces a new Proof of Deletion protocol, which assures secure data deletion in a way that any independent third party can verify cryptographically. The use of cryptography in data deletion is not new, but previous solutions are unsatisfactory as they merely return a single bit: whether the deletion is successful. However, this single bit is not easily verifiable. The implementation of the underlying deletion mechanism is entirely opaque to a user. This is particularly the case when the encryption program is encapsulated within a tamper resistant chip. Furthermore, all existing solutions claim only to make "best efforts" to delete data, but without any commitment to the outcome. Our protocol systematically addresses all these issues. First, we provide an auditing function to facilitate a user verifying that the encryption was done correctly. Second, instead of returning just one bit like all previous data deletion schemes, our solution returns a proof of deletion that is universally verifiable. This returned proof formalizes the commitment of the storage system in erasing data and cryptographically binds this commitment to the outcome of the operation.

About the authors

Feng Hao is a Lecturer of CSR. He received his BEng (2001) and MEng (2003) in electrical and electronic engineering from Nanyang Technological University, Singapore, and a PhD (2007) in computer science from the University of Cambridge, England. His research interests include biometrics, cryptography, fuzzy search algorithms, information coding, and error correction codes.

Dylan Clarke is a Research Associate in the School of Computing Science. He received his MMath in 2002, his MSc in SDIA in 2008 and his PhD in 2012, all from Newcastle University. Between 2003 and 2007 he worked for Redcar and Cleveland Borough Council developing CRM systems, and received an Innovation Award and a letter of commendation for this work. Dylan's PhD centred around the topic of enhancing intrusion resilience in publicly accessible distributed systems and was supervised by Dr. Paul Ezhilchelvan. His current research interests are e-voting, authenticated key exchange and other security topics.

Suggested keywords

SECURITY
DELETION
CRYPTOGRAPHY
TRUST
How to Delete a Secret

Feng Hao, Dylan Clarke
School of Computing Science
Newcastle University
{feng.hao, dylan.clarke}@ncl.ac.uk

Abstract. This paper discusses the secure data erasure problem and introduces a new Proof of Deletion protocol, which assures secure data deletion in a way that any independent third party can verify cryptographically. The use of cryptography in data deletion is not new, but previous solutions are unsatisfactory as they merely return a single bit: whether the deletion is successful. However, this single bit is not easily verifiable. The implementation of the underlying deletion mechanism is entirely opaque to a user. This is particularly the case when the encryption program is encapsulated within a tamper resistant chip. Furthermore, all existing solutions claim only to make “best efforts” to delete data, but without any commitment to the outcome. Our protocol systematically addresses all these issues. First, we provide an auditing function to facilitate a user verifying that the encryption was done correctly. Second, instead of returning just one bit like all previous data deletion schemes, our solution returns a proof of deletion that is universally verifiable. This returned proof formalizes the commitment of the storage system in erasing data and cryptographically binds this commitment to the outcome of the operation.

1 Introduction

You normally change the problem if you can’t solve it.
– David Wheeler

Alice saves her data on a disk storage system that uses magnetic drives. She routinely deletes some data, which may be confidential. When she deletes a secret, she wishes the deletion to be permanent. The disclosure of the secret may harm her privacy. The question is: how to ensure to delete a secret securely?

One simple solution is to physically destroy the disk. However, that would imply that every time when Alice wishes to delete some secret data, she would have to purchase a new disk. The cost will be prohibitively high.

There are two alternative solutions: through overwriting or encryption [5]. The first method works by using software means to overwrite disk locations where the data is stored. However, as explained in [4,10,12,14,16,18], an attacker with an advanced microscope and suitable tools may still be able to recover deleted data even after the disk has been overwritten many times. The second method
works by encrypting data using a key, and then deleting the data by disposing of
the key [5,15,16]. In this paper, we will focus on discussing the second method.

The encryption-based approach simplifies the problem of deleting a large
amount of data to deleting a short secret key (say 16 bytes), however the funda-
mental question remains to be answered: how to securely delete the key. Three
solutions were proposed in the past literature. The first is by physical destruc-
tion. As suggested by Boneh and Lipton [5], the user is asked to write down the
key on paper or a floppy disk, and later delete the key by physically destroying
the medium. However, this method can prove inconvenient and error-prone in
practice. The second method is through overwriting. As described in [12,16],
the key is stored on the disk alongside the ciphertext, and the deletion is done
by overwriting the location where the key is stored. However, as we explained
earlier, this returns to the previous argument that a clean deletion on the disk
is not possible [9]. The third approach is to use a tamper resistant module (e.g.
a smartcard) to store the key [15]. The module deletes the key by removing the
reference to the key from the program and overwrites the memory location where
the key was stored with zeros or random data.

All the above software-based deletion methods – through overwriting [4,10,
12,14,16,18] or encryption [5,12,15,16] – return just a single bit indicating
whether the deletion is successful. This however has two limitations. First, the
single bit is not easily verifiable. The user has to trust that the deletion software
had been implemented correctly in the first place. Second, the single bit does not
clearly indicate liability in the case of a failure. For example, a user may claim
that the program returned a “YES” bit, though the secret had not been been
deleted. Meanwhile, we cannot exclude the possibility that the user might have
misread the bit or might have dishonestly flipped its value (in order to discredit
the program). This ambiguous liability can lead to misplaced incentives. Instead
of focusing on ensuring a correct implementation of secure data erasure, the
solution provider may try to deny the existence of problem and claim that the
complaints come from dishonest users.

In this paper, we will propose a solution that addresses all of the above
problems. Our solution is also based on cryptography, but it differs from the
previous cryptography-based methods (see [5,15,16]) in two ways. First, instead
of returning just a single bit, it returns a proof that any independent third party
can verify. The proof is, as we will explain later, a digitally signed Service Level
Agreement. Second, we improve the transparency of the encryption process by
adding an auditing mechanism through which the user can verify whether the
encryption was done correctly. In contrast, in previous works [5,15,16], there is
no means for verifying the correctness of the encryption software. The fact that
the decryption of an encrypted message returns the same plaintext message is
insufficient to assure the encryption program had been implemented correctly.
In the following section, we will explain our solution in detail.
2 Our solution

Before describing the technical solution, there is a business obstacle we need to overcome. In the usual business model, data storage is a paid service (Alice pays to buy a disk for storing data), but the deletion is free (she can freely delete data to get more storage space). A free service naturally comes with no assurance. This model provides strong incentives for the storage providers to improve the reliability of the disk, but not much to assure secure data erasure.

To make the solution more meaningful, we slightly change the problem by assuming Alice is willing to pay for secure data erasure. We do not find this assumption explicitly made in previous papers [4,5,8–16,18], but we believe this is a reasonable assumption. Given that Alice pays for secure data erasure, it is natural for Alice to expect higher assurance than just a “best-effort operation”. The mere single bit return is clearly unsatisfactory.

2.1 System overview

We give an overview of a secure Data Storage and Erasure (DSE) system in Figures 1. The system consists of two basic components: a mass storage device and a tamper resistant module. For simplicity, we call the former a “disk” and the latter a “chip”. The user is able to freely access any location on the disk. The chip has an embedded processor and a secure memory. All crypto keys are kept inside its memory. A host program cannot directly access the keys but it can make use of the keys through specified Application Programming Interfaces (APIs). Normally, the host program must be authenticated to the chip before being able to invoke APIs (e.g., through passwords, user biometrics or some challenge-response protocol).

![Fig. 1. Overview of a secure Data Storage and Erasure (DSE) system](image)

2.2 Protocol

We now describe a Proof of Deletion (PoD) protocol in the setting of an additive cycle group defined over an elliptic curve. (Alternatively, we can also use
Notations | Meaning
--- | ---
Prv\(_{\text{chip}}\), Pub\(_{\text{chip}}\) | Unique ECDSA key pair for each chip
C | Client user
\(C_i\) | Instance of the client
Prv\(_{C_i}\) | Private key of the client instance, Prv\(_{C_i}\) := \(d_{C_i}\)
Pub\(_{C_i}\) | Public key of the client instance, Pub\(_{C_i}\) := \(d_{C_i} \cdot G\)
m | Input message
\(Q_\eta\) | Ephemeral public key during DHIES \(Q_\eta = d_\eta \cdot G\)
\(E_{\text{Auth}}(m)\) | Authenticated encryption of \(m\) using a symmetric key \(k_\eta\)
E(Pub\(_{C_i}\), m) | Encryption of \(m\) under Pub\(_{C_i}\) using DHIES, E(Pub\(_{C_i}\), m) := \(\{Q_\eta, E_{\text{Auth}}(m)\}\)
ZKP\(_\eta\) | Zero Knowledge Proof to prove the well-formedness of ciphertext \(\eta\)
SLA\(_{C_i}\) | Service Level Agreement for the deletion of client instance \(C_i\)
sig(...) | Signed message using the chip’s ECDSA private key Prv\(_{\text{chip}}\)

Table 1. Notations and meaning

a multiplicative cycle group of prime order; this does not make any difference to the basic protocol.) Let \(E\) be an underlying elliptic curve for the Elliptic Curve Digital Signature Algorithm (ECDSA) [17] and \(G\) be a base point on the curve with the prime order \(n\). Each chip contains a unique ECDSA signature key pair: Prv\(_{\text{chip}}\) and Pub\(_{\text{chip}}\) (which are normally generated on-board during the personalization stage at the factory so the private key never leaves the chip). The chip manufacturer shall publish the ECDSA public key for every chip on a public website. We define the following APIs for the chip (refer to Table 1 for a summary of notations).

- **KeyGen(1\(^k\), C)** creates an instance of the client user \(C\). It takes as input a security parameter \(1^k\) and the identity of the user \(C\), generates a private key on-board Prv\(_{C_i}\) := \(d_{C_i} \in [1, n-1]\), and returns the corresponding public key Pub\(_{C_i}\) := \(d_{C_i} \cdot G\) and an index reference \(C_i\) to the created key pair. The user \(C\) is free to create as many instances as she wishes, subject to the constraint of memory size. As an example, with 160-bit \(n\), 32-bit index \(C_i\) and a chip of 16 MB EEPROM memory (see [1]), up to 666,667 user instances can be created. This function can be formalized as (for simplicity, we will omit error returns in all functions):

\[
\text{Host} \rightarrow \text{Chip} : 1^k, C \quad (\text{KeyGen})
\]

\[
\text{Chip} : \text{Generate Prv}_{C_i} := d_{C_i}
\]

\[
\text{Chip} \rightarrow \text{Host} : \text{Pub}_{C_i} := d_{C_i} \cdot G, C_i
\]

- **Encrypt(C\(_i\), m)** takes as input the reference to the created user instance \(C_i\), a message \(m\) and returns the encrypted message under the public key Pub\(_{C_i}\). For the encryption, we adopt the Diffie-Hellman Integrated Encryption Scheme (DHIES) [3], which works as follows. First, the chip generates an ephemeral public key \(Q_\eta = d_\eta \cdot G\) where \(d_\eta \in [1, n-1]\). It then computes
\( k_\eta = H(d_{C_i} \cdot Q_\eta) \) where \( H \) is a cryptographic hash function. Subsequently, the symmetric key \( k_\eta \) is used to encrypt the message in an authenticated manner to obtain \( E_{k_\eta}^{\text{Auth}}(m) \). (The authenticated encryption involves splitting \( k_\eta \) into two halves, \( \text{macKey}_\eta \) and \( \text{encKey}_\eta \), with the first half used to encrypt the message and the second half to compute an authentication tag.) Details about DHIES can be found in [3]. The returned ciphertext will be stored in the mass storage device with a reference \( \eta \) (which may be a location address on the disk). This function can be formalized as:

\[
\text{Host} \rightarrow \text{Chip} : \quad C_i, m \quad (\text{Encrypt})
\]

\[\text{Chip} \rightarrow \text{User} : Q_\eta := d_\eta \cdot G, \ E_{k_\eta}^{\text{Auth}}(m)\]

- **Decrypt\((C_i, \eta)\)** takes as input the reference of an existential user instance \( C_i \) and a ciphertext referenced by \( \eta \) and returns a decrypted message. The function follows the decryption procedure as detailed in DHIES [3]. If the ciphertext has not been tampered with, the original message \( m \) will be returned. This function can be formalized as:

\[
\text{Host} \rightarrow \text{Chip} : \quad C_i, Q_\eta, E_{k_\eta}^{\text{Auth}}(m) \quad (\text{Decrypt})
\]

\[\text{Chip} \rightarrow \text{Host} : \quad m\]

- **Audit\((C_i, \eta)\)** takes the same input as in **Decrypt** and allows the user to verify whether the encryption was done correctly. The ciphertext consists of two components, \( Q_\eta \) and \( E_{k_\eta}^{\text{Auth}}(m) \), which are related by the formula: \( k_\eta = H(d_{C_i} \cdot Q_\eta) \). To allow the auditing of the first component, the chip checks that \( Q_\eta \) is a valid public key on the curve, and then outputs \( d_{C_i} \cdot Q_\eta \) and a Zero Knowledge Proof (ZKP), which proves that \( \log_G d_{C_i} \cdot G = \log_{Q_\eta} d_{C_i} \cdot Q_\eta \) without leaking anything about the private key \( d_{C_i} \). The ZKP is based on the Chaum-Pederson protocol [6], which is made non-interactive by applying the Fiat-Shamir transformation [7]. Subsequently, the host is able to compute the symmetric key \( k_\eta = H(d_{C_i} \cdot Q_\eta) \) and verify the ciphertext according to the specified symmetric cipher (say AES). Note that this auditing only reveals the session key for the encryption of a specified message; other messages encrypted under the same user instance’s public key are not affected. This function is formalized as:

\[
\text{Host} \rightarrow \text{Chip} : \quad C_i, Q_\eta, E_{k_\eta}^{\text{Auth}}(m) \quad (\text{Audit})
\]

\[\text{Chip} \rightarrow \text{Host} : \quad d_{C_i} \cdot Q_\eta, \ ZKP_\eta[\log_G d_{C_i} \cdot G = \log_{Q_\eta} d_{C_i} \cdot Q_\eta]\]

- **Delete\((C_i)\)** deletes a user instance \( C_i \) by overwriting its private key \( d_{C_i} \) in the chip’s memory and returns \( \text{SLA}_{\text{del}}^{C_i} \), which is a Service Level Agreement {"Delete", \( \text{Pub}_{C_i} \)} signed by the chip’s ECDSA signing key. After the erasure of the private key, all messages encrypted under \( \text{Pub}_{C_i} \) can no longer be decrypted. If the user discovers that the system failed to erase the private key as instructed, she can prove this in court by presenting \( d_{C_i} \) together with \( \text{SLA}_{\text{del}}^{C_i} \). Accordingly, she should be entitled to compensation on the grounds of the violation of the Service Level Agreement. This function is formalized as:
3 Analysis and discussion

**Data thief** A data thief is an adversary who wishes to recover the deleted secret. We assume the attacker has physical access to both the disk and the chip. He is able to read all data from the disk including those that had been overwritten or deleted in the past. In addition, he is able to bypass the authentication mechanism (e.g., he knows the password), and invoke the APIs of the chip. Under this threat model, our goal is to prevent him from recovering *deleted* secrets.

All data stored on the disk is encrypted under the DHIES algorithm. The correctness of the cipher implementation can be verified by the user through the Audit function. Based on the security proofs of the DHIES paper [3], without the private key, the ciphertext is indistinguishable from random data. Hence, the attacker must obtain the key.

To recover a deleted key, the attacker needs to penetrate two lines of defense. First, he needs to break the tamper resistance of the chip in order to access the memory. Second, he needs to recover the overwritten key from the memory. Breaking the tamper resistance is not impossible, but can prove costly. Furthermore, if the program on the chip did overwrite the key in the memory with random data even for one or two passes, recovering the key can prove very expensive [8, 9]. In summary, if the chip vendor has implemented the protocol correctly, the cost of recovering a deleted secret can be quite high for the attacker (probably higher than the value of the secret itself).

**Incompetent Vendor** The above analysis is based on the assumption that the chip vendor is competent to get the software correct, but that may not be case. Software bugs and implementation blunders can leave the chip in a vulnerable state, allowing an attacker to easily bypass the physical tamper resistance [2]. In that case, the cost of breaking the tamper resistance can be significantly lowered. In addition, if the software does not overwrite the key in memory (despite that this is a trivial step in the code), an attacker may further be able to extract the deleted key by scanning the memory. In the real world, if serious vulnerabilities of a “tamper resistant” chip are discovered, the stock price of the chip vendor will drop, and customers will choose a different supplier. Based on the arguments of security economics [2], a security system should be designed such that the entity in the best position to protect security must be liable for the failure. Our solution just does that.

Here, we do not consider the threat of a dishonest vendor. We assume the chip vendor is a commercial entity, delivering a solution to a mass market. It is in their commercial interest not to put malicious software on the chip. We understand this assumption may be challenged in some cases. But if a user deals with high-grade secrets, she probably should not use an off-the-shelf product. Instead, she should carefully choose a provider that she trusts (or in the extreme case implement her own solution).
**Dishonest user** A dishonest user is one who wishes to profit from claiming compensation. To claim for compensation, the user will need to present a signed SLA together with the private key $d_C$ that is supposed to have been deleted. There are two ways of attack: 1) by overcoming the tamper resistance and then extracting the ECDSA signing key from the chip memory; or 2) by overcoming tamper resistance and then recovering the overwritten key in the memory. If a chip is sufficiently costly to reverse engineer, a dishonest user will not be able to profit from this attack. On the other hand, if the chip is vulnerable, one will be able to prove that in a publicly verifiable way. The proof will indicate that either the chip has failed to protect the ECDSA private key with tamper resistance or it has failed to securely erase the key from the persistent memory. Between the two attacks, the first one can be easier than the second. As long as the dishonest user is able to extract the ECDSA private key from the secure memory, he does not need to go further to recover the overwritten key in the memory. He can use the extracted private key to generate an SLA, and demand compensation. This is a shortcut in the attack, which needs to be considered when devising the pricing strategy as we discuss below.

**Pricing strategy** If the user shows evidence to prove that the chip vendor has failed on the Service Level Agreement, she may be entitled to compensation. Depending on the details of SLA, the compensation may be a fixed amount or an amount that is proportional to what the user pays for buying the solution. We begin by taking $C_1$ to be the cost of penetrating the secure memory in the chip (i.e., so as to extract the ECDSA signing key) and $C_2$ to be the cost of recovering an overwritten key from the secure memory. Hence, the overall cost to recover a deleted key from the chip is: $C = C_1 + C_2$. We denote $P$ be the selling price of that chip, and $N$ be the amount of compensation. As long as $N \leq P + C_1$, then no dishonest user will be able to make a profit by reverse engineering a chip. Let us denote $T$ as the cost of producing the chip. When $P - T > N$, a chip vendor will be able to unfairly profit by simply producing low-cost but insecure chips and attempt to sell a large amount of them before this insecurity is discovered. Then, when the insecurity is discovered, the provider can pay compensation to every user, making a profit of $P - T - N$ for each chip. Hence, a reasonable amount for compensation should be defined within the following range:

$$P - T \leq N \leq P + C_1$$

### 4 Conclusion

In this paper, we have presented a Proof of Deletion (PoD) protocol, which ensures secure data erasure based on cryptography. As compared with related schemes in the past, our protocol improves transparency in the encryption by allowing a user to easily audit the encryption process. In addition, while previous solutions all chose to make a “best efforts” attempt at deleting data, our scheme does this but also provides a proof of commitment. In the case of failure, the
proof will serve as universally verifiable evidence to indicate failure, based on which the user should be entitled for compensation.

References