Making Decryption Accountable

Mark D. Ryan

University of Birmingham

Abstract. Decryption is accountable if the users that create ciphertexts can gain information about the circumstances of the decryptions that are later obtained. We describe a protocol that forces decryptors to create such information. The information can't be discarded or suppressed without detection. The protocol relies on a trusted hardware device. We describe some applications.

1 Introduction

When I was a teenager, I wanted to be able to go out in the evening and not have to tell my parents where I was going. My parents were understanding about this wish for privacy, but explained that if for some reason I didn't come back at the expected time, they needed to have some clues to give to the police about where I had been. So we came to the following compromise: I would leave a sealed envelope explaining my activities. This would enable them to search for me if they needed to, but if I came back on time I could retrieve the envelope and see that it had not been opened. This idea seems closely connected to the 'multi-objective' theme of SPW'17: the protocol aims to simultaneously serve the partly conflicting requirements of the teenager and his parents.

To have such a protocol in the digital world, we would need some way of knowing whether someone who has all the needed material to perform a decryption has actually performed it. More generally, we need a way to make decryption key holders accountable in some way for their use of the key. This accountability might take many forms. For example, some applications might need fine-grained accounts of exactly what was decrypted, and when, while in other cases we may be interested only in volumes, frequencies, or patterns of decryption.

In this paper, we informally describe the requirements for making decryptions accountable (section 2), and devise a protocol based on trusted hardware that achieves them (section 3). We describe a few applications at a very high level (section 4).

2 The requirements

We formulate the requirements as follows:

- Users $U_1, \ldots$ create ciphertexts using a public encryption key ek.
- Decrypting agent Y is capable of decrypting the ciphertexts without any help from the users.

Authors' preprint of 2015-06-30.
Final version to appear in *Proceedings of Passwords 2014*, Springer LNCS.

– When Y decrypts ciphertexts, it unavoidably creates evidence e that is accessible to the users. The evidence cannot be suppressed or discarded without detection.
– By examining e, the users gain some information about the nature of the decryptions being performed.

Here, the granularity of e is left open. We will see some examples in section 3.2.

Note that we focus on ensuring that if Y decrypts, then the user will be able to achieve evidence that that has happened. The teenager envelope story in the introduction had an additional property: the parents Y can give up the possibility of ever decrypting, if they wish (by returning the unopened envelope). We don't include that additional property in our requirements.

3 Protocol design

Intuitively, if Y has a ciphertext and a decryption key, it is impossible to detect whether she applies the key to to ciphertext or not. This implies that the key has to be guarded by some kind of hardware device D that controls its use. In this section, we propose a simple generic design that achieves some of the desired functionality. The hardware device D embodies the secret decryption key dk corresponding to ek. The secret decryption key dk never leaves the device.

In order to make the evidence e persistent, we assume a log L. The log is organised as an append-only Merkle tree as used in, for example, certificate transparency [1]. The log maintainer publishes the root tree hash H of L, and is capable of generating two kinds of proof about the log's behaviour:

– A proof of presence of some data in the log. More precisely, given some data d and a root tree hash H of the log, the log maintainer can produce a compact proof that d is indeed in the log represented by H.
– A proof of extension, that is, a proof that the log is maintained append-only. More precisely, given a previous root tree hash H' and the current one H, the log maintainer can produce a proof that the log represented by H is an append-only extension of the log represented by H'.

(Details of these proofs can be found in e.g. [8].) This means that the maintainer of L is not required to be trusted to maintain the log correctly. It can give proofs about its behaviour.

3.1 Performing decryptions

The decrypting agent Y uses the device D to perform decryptions. The device will perform decryptions only if it has a proof that the decryption request has been entered into the provably-append-only log.

The device maintains a variable containing its record of the most recent root tree hash H that it has seen of the log L. On receiving a set R of decryption requests, the decrypting agent performs the following actions:

- Obtain from the device its last-seen root tree hash H.
- Enter the set R of decryption requests into the log.
- Obtain the current root tree hash H' of the log.
- Obtain from the log a proof π of presence of R in the log with RTH H'.
- Obtain from the log a proof ρ that the log with RTH H' is an append-only extension of the log with RTH H.

The decrypting agent presents (R, H', π, ρ) to the device. The device verifies the proofs, and if they are valid, it performs the requested decryptions R. It updates its record H of the last-seen root tree hash with H'.

3.2 Evidence

Evidence about decryptions is obtained by inspecting the log, which contains the decryption requests. There are many ways that this could be organised. We look at two examples:

Example 1: the log contains a hash of the ciphertext that is decrypted. This allows a user U to detect if ciphertexts she produced have been decrypted.

Example 2: the log contains a unique value representing the decrypted ciphertext, but the value cannot be tied to a particular ciphertext (for example, the value could be the hash of a re-encryption [7]). This allows users to see the number of ciphertexts decrypted, but not which particular ones.

3.3 Currency

As described so far, the protocol is insecure because the device D could be tracking a version of the log which is different from the version that the users track. Although both the device and users verify proofs that the log is maintained append-only, there is no guarantee that it is the same log. The log maintainer can bifurcate the log, maintaining each branch independently but append-only.

Gossip protocols of the kind proposed for solving this problem for certificate transparency [5] are insufficient here, because the device D is not capable of reliably participating in them.

To ensure that users track the same version of the log that D tracks, we introduce an additional protocol of D. In this second protocol, D accepts as input a *verifiably current* value v. The value v cannot be predicted in advance, but is verifiable by anyone. D outputs its signature $\mathrm{Sign}_{sk}(v, H)$ on the value v and its current stored root tree hash H of the log. Thus, we require that D has an additional secret key sk for signing. The corresponding verification key vk is published. Like dk, the key sk never leaves the device.

There are several ways in which the verifiably current value v can be constructed. For example, v can be the hash of a data structure containing nonces $v_1, \ldots$, each one produced by one of the users $U_1, \ldots$. Alternatively, v could be the concatenation of the date and the day's closing value of an international stock exchange.

Periodically, the current value of H tracked by the device is published. By means of the proofs of extension, users can verify that it is consistent with their view of the log.

There remains the possibility that users can be denied the possibility of inspecting the log, and/or denied the possibility of interacting with the device to obtain the log root-tree hash H. In these cases, decryptions can take place without the user being aware; but of course, the user knows s/he is being denied access. Thus, if users are denied access they should assume that the agreement concerning accountability has broken down.

3.4 The trusted hardware device

The protocol described relies on having a trusted hardware device D that performs a specific set of operations that are recapped here. The aim is to keep the functionality of D as small and as simple as possible, while still allowing it to support the variety of applications mentioned below (section 4). In summary, D stores persistent keys dk (decryption) and sk (signing), and the current root tree hash H of a log. It offers two services:

Decryption. It accepts a tuple (R, H', π, ρ) as described in section 3.1. It verifies the proof π that R is present in the log with root tree hash (RTH) H', and the proof ρ that H' is the RTH of a log that is an extension of the log of which its current RTH is the H stored in D. (These verifications consist of some hash calculations and comparisons.) If the verifications succeed, it performs the decryptions R, and replaces its stored H with H'.

Attestation. It accepts a value v, and returns $\mathrm{Sign}_{sk}(v, H)$ on the value v and its current stored RTH H.

4 Applications

4.1 Application areas

Most electronic voting protocols begin with voting clients that encrypt votes with a public key, and end with the result being decrypted by a trustworthy party (or, possibly, a set of trustworthy parties each of which holds a share of the decryption key). The decrypting agents are trusted only to decrypt the result, and not the votes of individual voters. A protocol to make decryption accountable could help make this verifiable.

Finance is an area in which privacy and accountability are often required to be balanced. For this reason, the designers of Zerocash have introduced mechanisms which allow selective user tracing and coin tracing in a cryptocurrency [6]. The limitation of their approach is that authorities have to decide in advance of the relevant transactions which coins or which users they want to trace. This is inconvenient in practice: often, suspects only become clear after transactions have taken place. Making decryptions accountable is another technique which

could help obtain the desired combination of privacy and accountability, and would allow retrospective decryption.

The UK government has recently passed legislation allowing government agencies to access information about the communications of private citizens [2], in order to solve crimes. In an effort to provide some kind of accountability, there are stipulations in the law to ensure that the provisions of the act are used in ways that are necessary and proportionate to the crimes being addressed. A protocol that makes decryption accountable could make verifiable the quantity and perhaps the nature of decryptions [7].

Making decryptions accountable potentially addresses the problem of having to trust escrow holders, for example in identity-based encryption [4] and elsewhere [3].

4.2 Access control

The device and the protocols are desiged to guarantee just one thing: that if decryptions take place, this fact can be detected by the user. Of course, in any real application such as those given above, much more than that is needed. For example, consider an email application, in which authorities are allowed to decrypt emails, and users can detect the extent to which this has been done. Our protocols provide this detectability, but they do not provide other features that would be required to make such a system acceptable. For example, in what circumstances can decryptions be requested, and by whom? How are the decrypted mails to be treated? All these questions would have to be answered by access control mechanisms layered on top of the protocols detailed in this paper.

5 Conclusion

There seems to be a variety of circumstances in which making decryption accountable is attractive. This paper proposes the design of trusted hardware which would assist in this process.

The idea of the design is that the decrypting agent has no way to decrypt data without leaving evidence in the log, unless it can break the hardware device D. This raises the question of who manufactures the device, and how the relying parties (both users $U_1 \ldots$ and decrypting agents Y) can be assured that it will behave as specified. It depends on the sensitivity of the information being processed. One idea is that it is jointly manufactured by an international coalition of companies with a reputation they wish to maintain.

References

1. Certificate transparency. Available: `www.certificate-transparency.org` (2007)
2. Investigatory Powers Act. Available: `www.legislation.gov.uk/ukpga/2016/25/contents/enacted` (2016)

3. Abelson, H., Anderson, R., Bellovin, S.M., Benaloh, J., Blaze, M., Diffie, W., Gilmore, J., Green, M., Landau, S., Neumann, P.G., et al.: Keys under doormats: mandating insecurity by requiring government access to all data and communications. Journal of Cybersecurity (2015)
4. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Annual International Cryptology Conference. pp. 213–229 (2001)
5. Chuat, L., Szalachowski, P., Perrig, A., Laurie, B., Messeri, E.: Efficient gossip protocols for verifying the consistency of certificate logs. In: IEEE Conference on Communications and Network Security (CNS). pp. 415–423 (2015)
6. Garman, C., Green, M., Miers, I.: Accountable privacy for decentralized anonymous payments. IACR Cryptology ePrint Archive 2016, 61 (2016), http://eprint.iacr.org/2016/061
7. Jia Liu, Mark D. Ryan and Liqun Chen: Balancing Societal Security and Individual Privacy: Accountable Escrow System. In: CSF (2014)
8. Mark D. Ryan: Enhanced certificate transparency and end-to-end encrypted mail. In Network and Distributed System Security (NDSS) (2014)