Escrowed Data and the Digital Envelope

King Ables and Mark D. Ryan

University of Birmingham, UK

1 Introduction

1.1 Privacy vs. security

As computers continue to permeate all aspects of our lives, there is a growing tension between the requirements of *societal security* and *individual privacy*. Societal security encompasses all ways in which we try to make the world more secure, including transport security, financial security, infrastructure security, etc. A prime mechanism for achieving this security involves collecting quantities of data about individuals, for example via ISP logs, mobile phone logs, ticketing systems, and banking systems. As a result, massive databases about every aspect of our lives are being collected by organisations in all the major industrial sectors (financial, transport, retail, telecom, internet, and health care).

These data collection has an impact on individual privacy. The unprecedented **longevity**, **searchability**, and especially the **composability** from different sources of these records imply a radical reduction in the level of individual privacy we can expect to enjoy over the coming decades. Numerous reports have documented this impact and its detrimental consequences to society's well-being (e.g., in the UK, [3, 2]).

There is no easy solution to this problem, because the security uses of the data are too important to be denied. Their use in crime detection is an example. About 440,000 requests by the police, local authorities and other permitted organisations to monitor telephone calls, emails and text messages were made in a 15 month period in 2005-06 in the UK [3, pp.32-33]. The "Intercept Modernisation Programme" is a UK Government initiative to centralise electronic communications traffic data in the UK in a single database [5, 8]. In another example, the UK intelligence agencies MI5 and MI6 have have sought full automated access to Transport for London's 'Oyster' smartcard database [10]. Debate about balancing security and privacy is taking place at all levels of society [9, 3, 2, 4, 1, 7], and will likely continue for many more years.

1.2 Escrowed data

We propose *escrowed data* as an approach that may be capable of providing an appropriate balance between the requirements of individual privacy and societal security. Roughly speaking, data that is collected is held in escrow for a certain period. During that period, the data may be accessed by an authority in order to provide societal security, e.g., for the purposes of crime investigation. It is expected that a minority of the data needs to be accessed in this way, since most people don't commit crime. After the escrow period is over, the data can be destroyed.

Various kinds of conditions can be put on when and whether the data held in escrow can be accessed by the authorities. Such conditions are likely to vary considerably according to the nature of the data, and we don't consider them in this paper. Instead, we propose a mechanism under which, at the end of the escrow period, the subject of the data can obtain unforgeable evidence about whether the data has been accessed or not. We assume that this fact is sufficient to prevent the authority making unnecessary accesses.

Numerous technical problems need to be solved in order to make this work, including adaptation of the mechanisms by which data is collected and stored, and the ways in which it is used. The core of such a solution needs to provide the following properties:

- Data held in escrow can be accessed by the authority at any time during the escrow period. No cooperation by the subject of the data is required, and the subject is unable to detect whether an access has been made or not, until the end of the escrow period.

– At the end of the escrow period, the subject of the data is able to obtain evidence that says *either* that the data has been accessed; *or* that the data has not been accessed, and has now been destroyed.

This arrangement gives the authority all the power it needs in order to guarantee societal security, while at the same time giving individuals guarantees about their privacy most of the time.

Example 1. In transport charging (for example, London Oyster card, or road usage charging), the data about journeys is held in escrow for a period. In most circumstances, it is never accessed, but under given conditions law enforcement officers can open up data about individual journeys without alerting the individuals involved. After some time-window, still unopened data can no longer be opened, and individuals obtain verifiable evidence about what data has been opened up about them, and what data has been destroyed.

1.3 This paper: the digital envelope

In order to escrow data in the *physical world*, one can store it in a sealed tamper-evident envelope such that it can be opened, but once opened, cannot be resealed. In this paper we present the concept of the *digital envelope* which provides a digital analogue of the envelope in the physical world.

The digital envelope allows Alice to provide digital data to Bob in such a way that Bob has only one of two possible actions available to him:

– He can access the data without any further action from Alice.
– Alternatively, he can revoke his right to access the data, and in this case he is able to prove to Alice that he did not and cannot (any longer) access the data.

Intuitively, it is not possible to achieve this effect using cryptography alone. Alice can encrypt the data and send the ciphertext to Bob. But if she does not send the key as well, then Bob can't unprotect the data without further cooperation from Alice. If she does send the key at the same time as sending the data, then Bob is not ever able to demonstrate that he has not decrypted it. Even if he "returns" the ciphertext to Alice, she has no guarantee that he has not decrypted another copy.

In this paper, we present three mechanisms for achieving the digital envelope in a trusted computing context, and compare them. Section 2 is devoted to background information about trusted computing and the TPM. Section 3 contains our three implementations, and the comparisons. Section 4 considers some modifications of the TPM, and we draw conclusions in Section 5.

2 Background

The *Trusted Platform Module* (TPM) is a commodity chip present on most high-end laptops currently shipped by all the major manufacturers. Through over 100 function calls, it provides protected cryptographic operations to general purpose software that runs on the platform.

In this short version of the paper, we assume readers are knowledgeable about TPM functionality, including platform configuration registers, creation and management of encryption keys, attestation, measurement, identity, monotonic counters, and encrypted transport sessions. A description of all this functionality is available in the longer version of the paper available on the author's web page.

3 Three implementations of the digital envelope using the TPM

We present three different possible solutions for a digital envelope by using functionality of the Trusted Platform Module. Each solution assumes Bob has a "recipient" computer (his own or a server) containing a functioning TPM. The solutions have varying levels of requirements and thus, a varying level of limitations in usability and functionality. They each have advantages and disadvantages compared to the others. No single solution is clearly superior.

3.1 No software required

Somewhat surprisingly, the digital envelope can be implemented directly using the functionality of the TPM, without any trusted software. However, the implementation has some limitations. The idea is to bind the data using a TPM key locked to specific PCR values.

Implementation

Sealing the envelope Alice creates an encrypted transport session with Bob's TPM and uses it to extend a given PCR with a random nonce n that she has created. She keeps the value of n secret. The transport session is then closed.

Alice or Bob reads the value of the given PCR, finding it to be p, say, and creates a `TPM_KEY_BIND` key (sk, pk) on Bob's TPM, locked to the PCR value SHA1$(p||1)$. This means that the key can be used only if the value 1 is first extended into the PCR.

Alice encrypts her data with pk, and sends it to Bob. This protocol is illustrated in Figure 1.

Opening the envelope Bob can use `TPM_Extend` to extend 1 into the relevant PCR. He can then use `TPM_Unbind` to decrypt the datagram sent to him by Alice, in order to obtain the data.

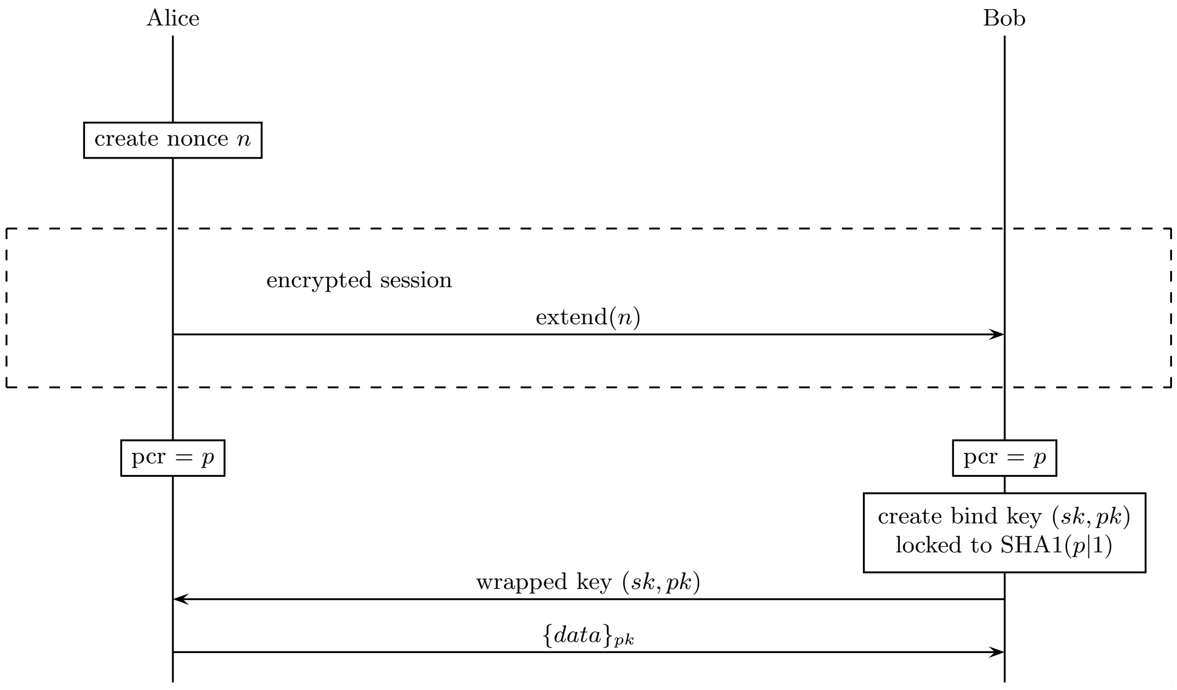

Fig. 1. Solution 1 (no software required): Alice sends envelope

Returning the envelope Alternatively, Bob can demonstrate that he has given up that possibility. To do that, he extends an agreed value, say 2, into the TPM. Alice may obtain a PCR quote to see that the value of the PCR is now SHA1$(p||2)$. This assures her that Bob can never use the key (sk, pk) to decrypt the datagram. This protocol is illustrated in Figure 2.

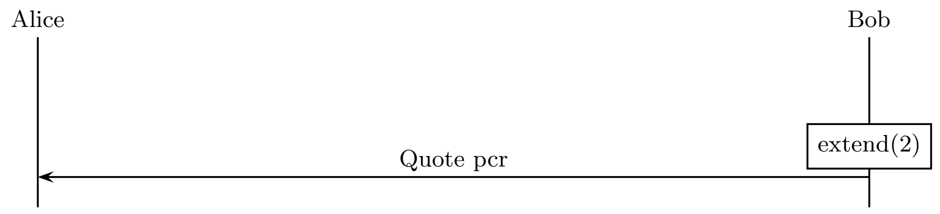

Fig. 2. Solution 1 (no software required): Bob returns envelope

Advantages and limitations The greatest advantage to this solution is it can be implemented on TPM platforms without requiring any trusted code on Bob's computer. It could be run on a user's personal system or a server system. Because the TPM controls access to the encrypted data, no application code requires trust or attestation.

The major disadvantage in using PCRs is that they maintain a volatile state which is lost when the TPM is reset, so this solution can only provide guarantees until the machine is rebooted (including after a crash). Once the system reboots, the PCRs will be reset to their default values and Bob will have lost both his ability to read the encrypted data as well as his ability to prove to Alice that he did not.

3.2 Attestation of envelope server code

Using the TPM and monotonic counters, a digital envelope mechanism can be created that can still be used when the system reboots. However, because monotonic counters are not used to seal the data, all of the code (including the operating system) processing the digital envelope must be attested to by the TPM so it can be trusted by Alice.

To reduce the complexity of the problem, we present a solution that is designed for a limited environment where it runs on a dedicated system. We assume that the system has a TPM and TCG-enabled BIOS and boot loader, and that the application runs native on the hardware with no operating system or virtual machine support. The digital envelope server is capable of processing only one envelope at a time. In addition to these specific assumptions about the platform, some way of obtaining PCR values for various makes and models of hardware platforms is also required.

Implementation To use the digital envelope server, Alice will create a blob containing the message that can only be opened by a TPM-verified digital envelope server. The procedure she will follow is:

- Request an envelope which includes a TPM-protected key tied to a monotonic counter value.
- Verify the envelope has been created by an authentic TPM running a properly installed and configured digital envelope server application.
- Tie the message to this key (i. e., insert the message into the envelope).
- Send the envelope to Bob.

Then, Bob can, at a time of his choosing, use the digital envelope server to open the envelope or obtain proof that he did not open it and forfeit his ability to ever open it. The act of opening or refusing the message increments the counter so neither operation can be repeated nor can the other operation be performed later.

The digital envelope server runs in two states: initialisation and service. The initialisation state, State 0, starts the service, creates or unwraps keys and data, and prepares to begin servicing envelope requests. The service state, State 1, is the "normal" operational state of the application.

State 0: initialisation State 0 initialises the environment in which the digital envelope server will run with the following steps:

– Unseal or create the initialisation blob containing the digital envelope server's AIK. If just created, seal the initialisation blob against current PCR values set by State 0.
– Load the digital envelope server AIK into the TPM, and advance to State 1 by extending a particular PCR.

The digital envelope server's *sealing key* can only be loaded during State 0 since it is sealed against the PCR values at the time the application is first executed. The sealing key requires no TPM AuthData because it is stored in a blob which was sealed against PCR values and is only accessible to a measurement-verified digital envelope server in State 0.

State 1: service In State 1, the digital envelope server waits to provide one of these services upon request:

– Create a new (empty) envelope, *or*
– Open an existing envelope and return the data, *or*
– Return proof of refusal to open an existing envelope.

When creating a new empty envelope, the digital envelope server returns the new public encryption key for the digital envelope, counter information, public digital envelope server AIK and certificate, and signed counter information to the envelope requestor.

Because the initialisation information is sealed against the PCRs at the time the system boots, once the digital envelope server has loaded its data, it extends a particular PCR to enter State 1 to guarantee that no other process may then access the initialisation information.

Operation When the system boots, the chain of trust is followed all the way to the digital envelope server. The first time the digital envelope server executes, it creates a sealing key using the command `TPM_CreateWrapKey` (with null AuthData) which will be used to seal the state information blob to State 0 (the current state). It also creates an AIK and random AuthData to be used with all envelope encryption keys using the TPM command `TPM_GetRandom`. The digital envelope server has the TPM sign a digest of PCA information to bind it to the public part of its AIK and submits this with its public EK to the PCA to obtain its AIK certificate. Lastly, the digital envelope server seals a blob containing its AIK, certificate, and a chosen monotonic counter name to the PCR value defining State 0. At this point, it can generate a `TPM_Quote` of PCR state signed with the digital envelope server AIK and then extend a particular PCR by 1 to advance to State 1.

Subsequent runs of the digital envelope server need only to restore the state, which will only succeed from State 0. This requires loading the digital envelope server sealing key, unsealing the initialisation state blob, loading the digital envelope server AIK into the TPM, generating a signed `TPM_Quote` of the PCR state, and extending a particular PCR to advance to State 1.

An envelope encryption key is created in State 1, so it is not protected by sealing against PCRs reflecting State 0. The envelope encryption key is protected by a random AuthData value created by the digital envelope server during initialisation and stored in the initialisation state blob. Because the initialisation state is protected by sealing against PCRs reflecting State 0, the AuthData is inaccessible to any other application at any other time. The AuthData for all envelope encryption keys is known only to the digital envelope server.

When Alice needs to send Bob data in a digital envelope, she generates a random nonce and sends the digital envelope server a request for a new envelope with the nonce.

The digital envelope server returns an "empty" digital envelope to Alice consisting of the digital envelope server AIK and certificate, the `TPM_Quote` of PCR values signed with the digital envelope server AIK, the name and new value of the incremented TPM monotonic counter, the public part of a new envelope encryption key (with AuthData known only to the digital envelope server), and her original nonce.

When Alice has verified the envelope and her nonce, she is ready to send her data to Bob. To do so, she will generate a random symmetric key and encrypt her message with it. She will

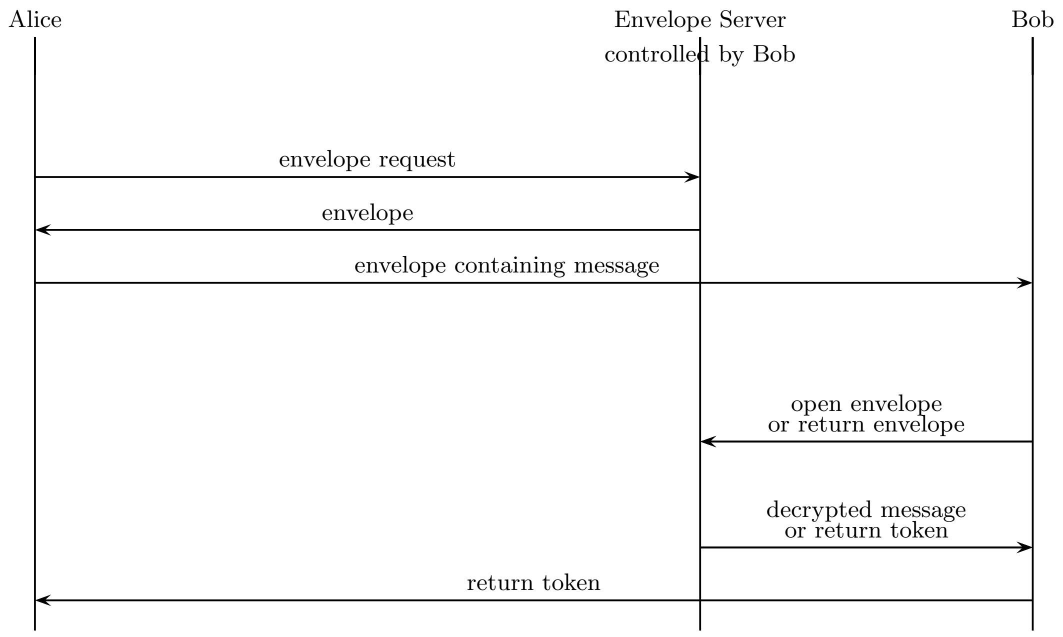

Fig. 3. Solution 2 (monotonic counter): the protocol

then use the public part of the envelope encryption key to encrypt the symmetric key and the other parts of the digital envelope. Alice can now send this data to Bob because only the digital envelope server can decrypt the contents of the digital envelope and will only do so if the named counter still has the specified value. Figure 3 shows the generic protocol for the digital envelope server.

Bob can now submit the digital envelope to the server for one of two purposes: to acquire the information sealed in the envelope or to obtain a token proving he has revoked his access. The digital envelope server validates the request, creates the response, increments the monotonic counter, and sends the response back to Bob. The digital envelope is no longer useful and Bob has either the data from Alice or a token he can send her to prove he did not and can no longer access it. The token may include Alice's nonce signed by the digital envelope server or a signed transport session showing the monotonic counter being incremented past the valid envelope value.

Advantages and limitations The main advantage of this implementation over the previous one is that it can save the envelope state across reboots of the platform. This comes at the cost of requiring trust in a small amount of software that manages the envelope software. Attestation is used to ensure the integrity of the software. No operating system is present.

The main limitation is that the platform is dedicated to providing the envelope service. Another disadvantage is the digital envelope server can only store and service one envelope at any time. Since each envelope requires its own counter and the TPM only allows the use of a single monotonic counter at any one time, *virtual monotonic counters* must be implemented to support multiple envelopes.

3.3 Flicker module

Flicker [6] is an infrastructure for executing TPM-attested code in isolation, while allowing a general purpose untrusted operating system with application software to run alongside it. Flicker is able to guarantee the attestation even if the BIOS, the operating system, and DMA-enabled devices are all untrusted. This is achieved by using hardware support for late launch DRTM, which features on high-end processors from AMD and Intel. Flicker works by causing the processor to temporally suspend the operating system, and to enter an attested configuration state where a small kernel, called a Flicker *piece of application logic* (PAL) is executed. The PAL is intended to run for a brief period, and return control to the operating system. It is hoped that the suspension time is short enough not to cause any unrecoverable disruption to the operating system. Before the end of its execution, the PAL is expected to save its state using the TPM's sealing functionality, and to recover its state at the beginning of the next execution. Flicker avoids replay attacks (in which the untrusted environment reverts to an old state of the PAL) by incorporating the current value of a monotonic counter into the saved PAL state, similarly to the way it is done in the previous subsection.

Implementation The Flicker implementation follows the pattern described for Flicker PALs that save state [6, §6.2]. The pattern focuses on maintaining the integrity of the PAL's state while the untrusted OS operates. To achieve this, the very first invocation of the PAL generates a 160-bit symmetric key based on randomness obtained from the TPM and uses the TPM to seal the key so that no other code can access it. It then performs application specific work. Before yielding control back to the untrusted OS, the PAL computes a cryptographic MAC (HMAC) over its current state. Each subsequent invocation of the PAL unseals the symmetric key and checks the MAC on its state before beginning application-specific work. When the PAL finally finishes its work unit, it extends the results into PCR 17 and exits.

The envelope-specific details are as described in the previous subsection (section 3.2).

Advantages and limitations The advantage over the previous implementation (section 3.2) is that the platform does not have to be dedicated to providing the envelope server. It can run a general purpose OS and applications. The cost of this is that the quantity of attested software is greater than that for the previous implementation, because Flicker adds a small additional overhead.

On the negative side, Flicker is currently experimental and has onerous software and hardware requirements, as well as dependency on the underlying processor architecture. These disadvantages can be expected to reduce over time, if continued development of the hardware technologies and of Flicker are made.

4 Suggested TPM enhancement: sealing to monotonic counters

The ability to seal data to monotonic counters (as well as to PCR values) would allow a significantly improved solution having the simplicity of our first solution (section 3.1) and the flexibility of our second (section 3.2). Such a solution could allow untrusted software to save the envelope state, and the TPM could detect replays that attempt to revert to a previous state.

4.1 New TPM commands

This can be achieved by providing the following proposed TPM commands.

– `TPM_SealByCounter` (*key*, *authdata*, *data-to-be-sealed*, *counter-name*, *counter-value*, *increment-on-unseal*)
  Here, *counter-name*, *counter-value* and *increment-on-unseal* may be represented in a `TPM_COUNTER_INFO` structure, analogous to the `TPM_PCR_INFO` structure used in the existing seal and unseal operations. This command seals arbitrary data with the specified key against a counter name

and value just as `TPM_Seal` seals against one or more PCR values. The *increment-on-unseal* is a boolean value which specifies whether or not the specified counter should be incremented when the data is unsealed.

– `TPM_UnsealByCounter` (*key*, *authdata*, *data-to-be-unsealed*)
  This command obtains the counter name and value from the blob and compares them to the current value of the named TPM counter. If they match, the TPM unseals the data. Upon successful unsealing of the data, but before it is returned to the caller, the named counter is incremented if *increment-on-unseal* was set to `TRUE` when the data was sealed.

4.2 No software required, v.2

Using these proposed new TPM commands, the digital envelope could be designed in a much more straightforward manner and could run within an unmeasured and untrusted application under any operating system.

Alice could request an envelope from the desired destination and receive the AIK and certificate as before, a signed log of a transport session proving the current monotonic counter and newly incremented value, two public parts of RSA key pairs, a signed log of a transport session showing these keys being created and sealed against the current monotonic counter (with *increment-on-unseal = TRUE*), and a TPM-signed copy of her nonce.

Alice can verify the envelope and that the keys were sealed properly against the proper counter value. She then encrypts her symmetric key with one public key and a refusal token with the other, encrypts her message with her symmetric key, and sends it all to Bob.

Using `TPM_UnsealByCounter`, Bob can ask his TPM to unseal either of the two envelope keys, but not both. Unsealing either the symmetric key for the message or the refusal token will cause the TPM to increment the monotonic counter which will eliminate the option to ever unseal the other.

5 Conclusion

We have presented the idea of a digital envelope that can provide data escrow in such a way that parties can obtain evidence about whether the data was accessed or not. This idea is expected to have applications in privacy management, and in particular to balancing the often conflicting requirements of individual privacy with societal security.

The Trusted Platform Module provides the primitives necessary to implement a digital envelope in a variety of ways. But the more straightforward the implementation, the more restrictive the functionality. As functionality is expanded to improve usability, security complications increase dramatically.

Due to the rigours of platform attestation, even the simplest solution quickly becomes complex. An additional capability like Flicker significantly minimises the impact of these additional issues.

It has also been shown that a much more straightforward solution could be achieved if the TPM provided a sealing operation using a monotonic counter analogous to the sealing operation it currently provides using Platform Configuration Registers. Therefore, two new TPM commands were proposed for addition to the TPM specification. If the TPM provided these commands in a future version, the measurement and attestation requirement would be eliminated and the digital envelope could easily be implemented in unmeasured code running on any operating system.

References

1. Petition to the UK Prime Minister against the email monitor database. `petitions.number10.gov.uk/privacy-matters/`.
2. K. Ball, D. Lyon, D. M. Wood, C. Norris, and C. Raab. A Report on the Surveillance Society for the Information Commissioner. `www.ico.gov.uk`, September 2006.

3. G. Crossman, H. Kitchen, R. Kuna, M. Skrein, and J. Russell. Overlooked: Surveillance and personal privacy in modern Britain. Published by Liberty. `www.liberty-human-rights.org.uk`, October 2007.
4. A. C. Grayling. Privacy is a quaint construct in a hyper-connected world. `www.guardian.co.uk/commentisfree/2008/dec/05/humanrights-privacy`.
5. H. M. Government. The united kingdom security & counter-terrorism science & innovation strategy. `security.homeoffice.gov.uk/news-publications/publication-search/general/science-innovation-strategy1`, 2007.
6. J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An execution infrastructure for tcb minimization. In *Proceedings of the ACM European Conference in Computer Systems (EuroSys)*, Apr. 2008.
7. I. of Eduction. Convention on modern liberty, 28 february 2009. `www.modernliberty.net/programme`.
8. Open Rights Group. Intercept modernisation. `www.openrightsgroup.org/orgwiki/index.php/Intercept_Modernisation`, 2009.
9. Privacy Blog. `www.theprivacyblog.com`.
10. The Register. Spooks want to go fishing in Oyster database. `www.theregister.co.uk/2008/03/17/spooks_want_oyster/`.