Public Key Pinning for TLS Using a Trust on First Use Model

Gabor X Toth[*] Tjebbe Vlieg[*]

[*]University of Amsterdam

February 2013

Abstract

Although the Public Key Infrastructure (PKI) using X.509 is meant to prevent the occurrence of man-in-the-middle attacks on TLS, there are still situations in which such attacks are possible due to the large number of Certification Authorities (CA) that has to be trusted. Recent incidents involving CA compromises, which lead to issuance of rogue certificates indicate the weakness of the PKI model. Recently various public key pinning protocols – such as DANE or TACK – have been proposed to thwart man-in-the-middle attacks on TLS connections. It will take a longer time, however, until any of these protocols reach wide deployment. We present an approach intended as an interim solution to bridge this gap and provide protection for connections to servers not yet using a pinning protocol. The presented method is based on public key pinning with a trust on first use model, and can be combined with existing notary approaches as well.

Keywords: TLS, PKI, X.509, public key pinning, trust on first use

1 Introduction

There are widely known problems with the Public Key Infrastructure using X.509 (PKIX) and its use in TLS certificate validation. Currently, users have to trust a large number of certificate authorities (CA), which are less and less trustworthy due to security breaches in their systems, leading to the issuance of rogue certificates [1, 2]. Even if a CA is not compromised, it can still issue intermediate CA certificates to the wrong entities [3], or it can be forced to issue rogue certificates by a government trying to spy on its citizens. [4]

Rogue certificates can be used in man-in-the-middle (MitM) attacks, which cannot be detected by only using PKIX certification path validation. For protection against such attacks additional certificate verification mechanisms should be put in place. In Section 2 we describe proposed solutions for this purpose, among them pinning protocols, which are supposed to provide a long-term solution to this problem. The adoption of these pinning protocols are not going to be instant, however, and could take a considerable time until they get deployed, as both client and server support are needed for them to work.

The trust on first use (TOFU) – or trust upon first contact – model, however, can be implemented client-side only and thus can be adopted rapidly. It is widely known for its use with SSH, it has also been proposed for use with e-mail authentication [5], and various other uses – including TLS server authentication – have been explored under the name Key Continuity Management [6]. In TOFU, a peer's credentials – public key, certificate, or the hash of one of these – is stored locally ("pinned") and is used for validation: a peer is expected to present the same credentials in subsequent connections, otherwise validation fails, in which case the user can choose between aborting the connection or updating the stored credentials if the change was legitimate.

In Section 3 we describe a public key pinning scheme for TLS server authentication based on a TOFU model, to be used as a fallback mechanism in case there is no pinning protocol available when connecting to a server. We also provide a proof of concept of this functionality implemented as a C library[1].

2 Related work

Several different approaches have been proposed to replace or improve the problematic X.509 PKI model for TLS server authentication. In this section we briefly describe these.

[1]Available at `http://tg-x.net/code/certpatrol`

2.1 Trust on first use model

As described in the introduction, in the TOFU model server credentials are pinned the first time a client establishes connection to a server.

Certificate Patrol [7] is a Firefox add-on that implements a TOFU model for certificate pinning, and notifies the user when a certificate change has occurred for a site. It shows the differences between the old and new certificates and tries to aid the user in figuring out whether it was a legitimate change by looking at whether the issuer has changed or the certificate was expired and had to be replaced.

2.2 Notary approaches

Notary services follow a different strategy: if a client sees a certificate for the first time, it is validated with the help of notary servers, which probe the server certificate from different network locations, making it much harder for an attacker to perform a MitM attack, as it would not only have to stand in between client and server, but also in between a majority of notary servers and the target server.

Perspectives [8] checks the certificate with one or more groups of notary servers of the user's choice, which monitor the public key of network services and store it in a database, thereby completely distrusting the current X.509 PKI model. It is implemented as a Firefox add-on.

Convergence [9] is similar to Perspectives, but extends the notary servers with different strategies to determine the correct certificate or public key for a service. It is also provided as a Firefox add-on.

Crossbear [10] also uses a notary approach, but tries to locate the man-in-the-middle (MitM) attacker as well by performing traceroutes from many different locations. It is not intended as a replacement for the current PKI system, but as a tool to identify and locate MitM attackers. Crossbear supports TLS and SSH. TLS certificate verification is implemented as a Firefox add-on, while the SSH version is provided as a patch for OpenSSH.

An issue with the notary approach is that users need to query notary servers regularly to verify certificates, which might expose the browsing history and habits of users, although some notary approaches – Convergence in particular – try to provide some level of anonymity for the users.

A scalability issue also exists with regards to notary servers, they need to be able to serve a large amount of users in case they would ever get popular.

This is also one of the main reasons Convergence is not included in Chromium by default [11].

2.3 Pinning protocols

Pinning protocols provide a way of establishing a specific – pinned – public key or certificate, which must be always present in the certificate chain sent by the server during TLS handshake. The pinned public key or certificate is actively designated by server administrators in this case. This model can be used as an alternative to the PKI model, but as both clients and servers need to support it, their deployment would take more time.

DNS-Based Authentication of Named Entities (DANE) [12] extends DNS with a new resource record to indicate a valid certificate or public key for a service using TLS. DANE checking functionality is provided by the libunbound library, and Firefox add-ons also exist, which are using this library. DANE is a proposed standard described in RFC 6698, and already has some – though still limited – deployment.

Trust Assertions for Certificate Keys (TACK) [13] is a TLS protocol extension that provides pinning of the public key of the certificate for a period of time, by signing it with a TACK key. Clients would check if the TACK key is the same as they have last seen it, instead of relying on the X.509 PKI. This approach requires changes to both TLS server and client implementations. The specification is in draft stage.

Public Key Pinning Extension for HTTP (websec-key-pinning) [14] employs HTTP headers to pin the public key of a certificate in the chain for a given period of time. As it relies on the HTTP protocol, it has a fairly limited scope unlike the other approaches, which improve TLS certificate verification in general, regardless of the application layer protocol. Its specification is in draft stage.

Monkeysphere [15] uses the OpenPGP web of trust (WoT) to store public keys of services. It supports SSH and TLS. The TLS version is implemented as a Firefox add-on and is intended to enhance the X.509 PKI by only verifying self-signed certificates using the OpenPGP WoT.

Most of the tools implementing these enhanced certificate verification methods described above are only available as a web browser add-on, while there are many more applications that rely on TLS to establish secure connections. A notable exception is DANE, for which libraries providing verification functionality already exist, but there are hardly any applications using it already.

3 Design

Public key pinning significantly enhances the security of TLS connections by specifying exactly which keys are valid for a server, instead of relying on the X.509 PKI. However, any of the proposed pinning protocols would take years to reach wide deployment.

An alternative pinning scheme based on a TOFU approach, which requires only client-side support, would enhance the practical security of certificate verification in TLS client applications. This would facilitate bridging the gap until a pinning protocol gets widely deployed.

In this section we discuss the design and requirements for such an approach, and its implementation on GNU/Linux systems.

3.1 Requirements for public key pinning

During the TLS handshake the server sends its certificate. Most of the time this is a X.509 certificate chain, but recently the use of raw public keys has been proposed too [16], in which case only a Subject Public Key Info structure is sent instead of a full X.509 certificate. Raw public keys are intended to be used with alternative validation methods provided by pinning protocols. We focus on the X.509 certificate type – supporting raw public keys would be just a special case of this, but most likely a pinning protocol would be in place anyway when they are used. The OpenPGP certificate type is not considered here, which is intended to be verified using a web of trust.

When there is no pinning protocol available for a server, a local pinning database should be consulted as a fallback mechanism, which is built according to a TOFU model: upon connection establishment the public key of a TLS server is stored, then later during a subsequent session the public key received is compared with the one stored locally. The problematic aspect of the TOFU model is that the client cannot be sure about the currently valid certificate used by the server, it trusts the first public key it received. If this pinned key changes later, the user is notified about the change and needs to decide whether or not to accept the change. To further complicate things, some larger sites running multiple servers behind a load balancer use multiple valid X.509 certificates for a single host name, which needs to be handled by allowing multiple pinned public keys per peer.

In case of X.509 certificates the raw Subject Public Key Info (SPKI) structure in DER format should be pinned, which contains the type of public key and data specific to that public key type. Pinning just the public key without the type would allow attacks using a different type of public key, while pinning the whole certificate would result in more change notifications then necessary.

This is because certificates are sometimes re-issued using the same public key, e.g. with a different hash algorithm for signature [17] – a pinned public key would thus remain valid when this happens. Storing and comparing the raw SPKI structure instead of using a hash of it has the advantage that it is faster and does not add dependency on the security of a hashing algorithm – many pinning protocols use hashing to reduce network traffic, which is not applicable in this case.

The Certificate Patrol Firefox add-on makes an attempt to tackle this problem, but in practice it can be quite obtrusive, requiring much user attention. This is mainly due to the fact that it always pins the end entity certificate, the issuer can be only pinned manually on a per-host basis, furthermore it only supports one active pin per host.

To make the use of TOFU with TLS more usable, pinning any public key in the certificate chain should be possible, this allows users to choose what to pin by default. While pinning the end entity certificate would be the most secure, it that would result in the most amount of change notifications as well. Pinning public keys higher up in the chain reduces the amount of notifications that require user attention. Storing multiple pins per peer also reduces the amount of notifications when a server uses multiple certificates.

The User Interface Guidelines published by the W3C [18] should be followed when presenting security notifications to the user. It is important to minimize the amount of notifications shown, otherwise an important change might go unnoticed. Using different severity levels for the notifications further helps to emphasize important messages.

3.1.1 Privacy concerns

Over time, visited host names with timestamps get stored in the pinning database. A functionality is thus needed to clear this history for a certain time period, similar to the "clear history" dialog provided by web browsers. Most web browsers also provide a private browsing mode, for which period no history should be saved, but public key pinning functionality is still desired. For this reason, during the private browsing session previously pinned keys need to remain accessible, but no changes should be made to the pinning database. Any changes should be written to a temporary in-memory storage. When reading, first this in-memory database is consulted, falling back to the read-only storage on disk.

3.2 Storage model

Pinning certificates using a TOFU model requires a local database, which associates peers with one or more public keys. A peer is defined by its host name, transport protocol – TCP in most cases, or UDP in case of Datagram TLS (DTLS) – and port number. The following fields are stored for each entry of a peer:

status	Status of the pin: inactive / active / rejected.
public key	The pinned SubjectPublicKeyInfo structure in DER format.
chain	The complete X.509 certificate chain for the peer in DER format. Empty when the peer uses a raw public key.
first seen	The first time this certificate was seen for this peer.
last seen	The first time this certificate was seen for this peer.
seen count	Number of times this certificate was seen for this peer.

When a certificate not yet in the database is received from a peer, a new entry is added, with its *status* initially set to *inactive*. The certificate chain received from the peer – completed from a local CA store in case of partial chain – is stored in the *chain* field, and the current timestamp is noted in the *first seen* field.

When a client receives a certificate that is already in the database, the *last seen* field is updated with the current timestamp, and the *seen count* field is incremented, but this is considered optional functionality that can be turned off when less logging is desired.

These values provide more insight when it comes to decide whether or not a certificate change could be legitimate. Storing the *chain* enables us to compare the issuing CAs of changed certificates, and to choose a different pinned public key later from the chain. The *seen count* and timestamps help determining whether a change is due to the concurrent use of multiple valid certificates by the peer.

Storing every certificate seen for a peer also helps identifying eventual attackers. In the event of an attack, the certificate used by the attacker is valuable information in the investigation. Integrating Crossbear as described in Section 3.4 would also help identifying and locating attackers.

Requirements for the storage are concurrent read and write operations by many processes of the same user. Suitable technologies for this purpose include dconf and SQLite, both of which support concurrent access. Dconf is specifically designed to handle frequent read operations efficiently, and supports layered databases, which can be used to provide read-only pinning information for all users on the system, with writable per-user pinning databases layered on top. This way user configuration takes precedence over system configuration, and trust decisions of different users do not affect each other, but an administrator or system builder can pre-configure a known set of pinned public keys so that users would not have to start from scratch. Such a list of pinned public keys is already included in the Chromium browser to protect well-known sites.

3.3 Verification process

During the TLS handshake the server sends its certificate, which has to be verified by the client. Algorithm 1 describes an enhanced verification algorithm that uses available pinning protocols with fallback to a TOFU pinning scheme together with PKI-based verification.

Using pinning this way ensures that blacklists are respected and the X.509 certificate chain is validated up to the specified trust anchor. PKIX certification path validation, which is described in [19]. If a pinned certificate later expires or gets blacklisted, an error would be shown, exactly like when no pinning is in use. When such an error occurs, the user might decide to override it by adding an entry to the override database, which associates peers with ignored error flags.

Lookup in the local TOFU pinning database happens according to the following: the verification process goes through the *active* or *rejected* entries stored for the peer, and stops when the *public key* field in an entry matches the raw DER-encoded Subject Public Key Info structure of one of the certificates in the chain being verified. The matched certificate is then used as trust anchor in case of an *active* entry, or part of the blacklist in case of a *rejected* entry.

The result of TOFU pinning verification is one of the following:

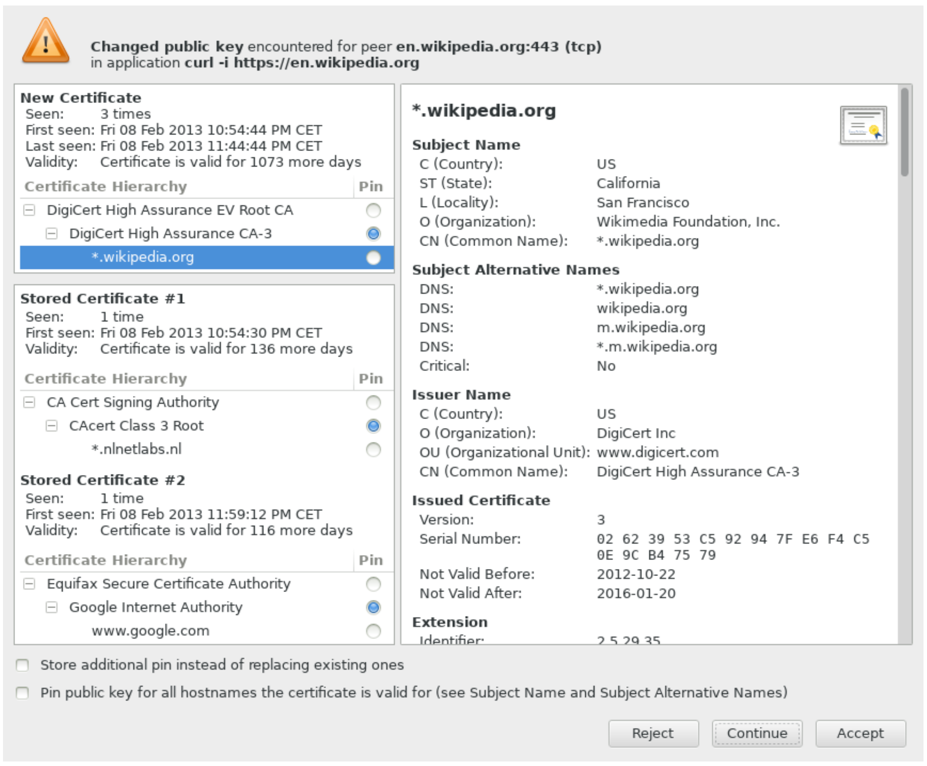
new	No *active* entry exists yet for the peer.
changed	No matching public key was found in any existing *active* entry.
rejected	A matching public key was found in a *rejected* entry.
OK	A matching public key was found in an *active* entry.

3.4 Notifications

Acting upon the result of verification is the following step. In case of a verification failure, either an error message that interrupts the user's task, a non-interrupting notification, or no notification is presented to the user, depending on the error condition and configuration. For instance web browsers display an interrupting error message when the certificate of a HTTPS document fails to verify. However, when the failure affects an included

```
Data: X.509 certificate chain sent by peer
Result: success or failure
if a pinning protocol is available for peer then
    use trust anchors supplied by pinning protocol
else if TOFU pinning database contains active entries for peer then
    use trust anchors specified by the active entries
else
    use system trust anchors
end
construct blacklist from system sources (e.g. CRLs, OCSP, local blacklists)
if TOFU pinning database contains rejected entries for peer then
    add the set of rejected public keys to blacklist
end
do PKIX path validation using trust anchors and blacklist set above as input
if validation returns with error then
    retrieve error flags that can be ignored for peer from override database
    if there are still error flags left after removing ignored ones then
        return failure
    end
end
return success
```

Algorithm 1: Certificate chain verification



Changed public key encountered for peer **en.wikipedia.org:443 (tcp)**
in application **curl -i https://en.wikipedia.org**

New Certificate
Seen: 3 times
First seen: Fri 08 Feb 2013 10:54:44 PM CET
Last seen: Fri 08 Feb 2013 11:44:44 PM CET
Validity: Certificate is valid for 1073 more days

Certificate Hierarchy	Pin
DigiCert High Assurance EV Root CA	○
DigiCert High Assurance CA-3	●
*.wikipedia.org	○

Stored Certificate #1
Seen: 1 time
First seen: Fri 08 Feb 2013 10:54:30 PM CET
Validity: Certificate is valid for 136 more days

Certificate Hierarchy	Pin
CA Cert Signing Authority	○
CAcert Class 3 Root	●
*.nlnetlabs.nl	○

Stored Certificate #2
Seen: 1 time
First seen: Fri 08 Feb 2013 11:59:12 PM CET
Validity: Certificate is valid for 116 more days

Certificate Hierarchy	Pin
Equifax Secure Certificate Authority	○
Google Internet Authority	●
www.google.com	○

***.wikipedia.org**

Subject Name

C (Country):	US
ST (State):	California
L (Locality):	San Francisco
O (Organization):	Wikimedia Foundation, Inc.
CN (Common Name):	*.wikipedia.org

Subject Alternative Names

DNS:	*.wikipedia.org
DNS:	wikipedia.org
DNS:	m.wikipedia.org
DNS:	*.m.wikipedia.org
Critical:	No

Issuer Name

C (Country):	US
O (Organization):	DigiCert Inc
OU (Organizational Unit):	www.digicert.com
CN (Common Name):	DigiCert High Assurance CA-3

Issued Certificate

Version:	3
Serial Number:	02 62 39 53 C5 92 94 7F E6 F4 C5 0E 9C B4 75 79
Not Valid Before:	2012-10-22
Not Valid After:	2016-01-20

Extension
Identifier: 2.5.29.35

☐ Store additional pin instead of replacing existing ones
☐ Pin public key for all hostnames the certificate is valid for (see Subject Name and Subject Alternative Names)

Figure 1: Changed public key notification dialog

resource instead – an image, style sheet, or script – browser behavior is quite diverse: from ignoring the error and loading the resource anyway, through silently skipping the resource, to displaying a dialog with an error message.

With TOFU pinning in use, a validation failure can occur due to a certificate change. Users should be always notified when this occurs, so that they can either accept or reject the change, both of which can have a permanent or just a temporary, one-time effect:

Accept temporarily Continue with the connection.

Reject temporarily Abort the connection.

Accept permanently Pin public key at the chosen level. This marks the current entry as active in the pinning database, and deactivates the other entries for the peer unless adding an additional pin is specifically requested.

Reject permanently Mark current entry as rejected, which results in suppressing further notifications about the certificate in question. This is particularly useful in case of included resources.

When a *new* certificate is encountered, a new pinned public key is added in the background: it is extracted from the certificate with the configured index in the chain – the *pin level* setting. Users who want more control can enable a notification – or even a dialog – when this happens.

The dialog – such as the one in Figure 1 – shows information about the current certificate in use by the peer, and the active certificates stored in the pinning database for the peer. For each entry the full certificate chain is displayed, which is the most important factor when deciding whether or not the change was legitimate: a MitM attacker would typically use a rogue certificate issued by a different CA other than the one used by the server, which makes this a good indicator. Also displayed are the *first seen* , *last seen*, and *seen count* values, which help identifying certificates already seen in the past, which could be indicating that the site is using multiple certificates simultaneously. The certificate expiry date is shown as well: often a certificate is exchanged soon before its expiry date, but this should not be a main factor in deciding whether or not the change was legitimate: on the one hand there are several legitimate reasons for exchanging a certificate before its expiry, on the other hand an adversary might decide to perform a MitM attack soon before a certificate expires in order for the change to seem more feasible. A scoring algorithm for comparing certificates – as used by Certificate Patrol and Crossbear – could be used here to provide an indication for the probability of a legitimate change.

At this point additional information from various external sources provided by notary services – such as Convergence or Crossbear – can be used if the user is still unsure about the change, which could be queried either manually or automatically, depending on configuration. Notary clients should be available as stand-alone applications in order for this to work – currently they are implemented as browser add-ons.

3.4.1 Logging

The interactive notifications described above are only suitable for desktop systems. In case of servers establishing TLS connections – e.g. an SMTP server sending out email – validation failures should be written to a log file instead. This would still require regular intervention of a system administrator to review and act on changed certificates, just like in the desktop system case, but asynchronously.

3.5 Integrating pinning functionality

Applications verify certificates using functionality provided by TLS libraries, of which OpenSSL, GnuTLS and NSS are the most popular ones. GLib and Qt provide an abstraction layer above TLS libraries, many applications use these instead of interfacing with a TLS library directly. GNOME applications use GLib, while those for KDE employ Qt. Mozilla software and the Chromium browser use NSS.

A problem of the diversity of TLS implementations is that there is no single database for trusted and blacklisted X.509 certificates and certificate authorities: different libraries and applications use different methods and formats to represent this information. A solution for sharing trust policy between different TLS libraries is desired in order to achieve consistent TLS certificate verification decisions across the system. One way to achieve this is using Stapled Certificate Extensions [20] – a proposal currently being worked on by the p11-glue project – which requires no changes to existing verification algorithms.

Implementing pinning functionality – using pinning protocols with fallback to TOFU pinning, as described above – as part of the verification process is also desired. This requires changes to the verification process in any case, sharing only a trust database is not enough for this to work. A common implementation of TLS certificate validation provided as a library that can be used by any application would solve this problem, and would also help improving the often incorrect use of validation functionality provided by current implementations – this latter aspect is described in [21]. Accomplishing this, however, requires consensus among libraries and applications using certificate validation functionality.

When such a common validation library is in place, it would become the source of trust anchors, blacklists, overridden validation errors, and pinning information for the entire system. This library would then need to provide a high-level certificate verification function that implements pinning functionality as well.

Libpkix – part of the NSS project – for instance already provides extensive validation functionality, which would at least be used by Firefox and Chromium, but it is not limited to applications using NSS to establish TLS connections. Using an external validation library together with any TLS implementation is straightforward: the DER-encoded X.509 certificate chain, as sent by the peer, needs to be passed to the validation library, which just involves placing it in the right data structure required by the library.

A perhaps less ambitious approach is to provide just the pinning functionality as a separate library, which should then be used together with certificate validation functionality of TLS libraries.

4 Conclusion

We have described a method of verifying TLS certificates using a trust on first use model, and how this can be used together with verification employing the X.509 PKI, as well as notary services. The result is a mechanism that can be used to effectively thwart man-in-the-middle attacks that the X.509 PKI-based validation would not detect. A longer term solution to this problem would be the use of pinning protocols – such as DANE or TACK – which are still being defined and implemented, and would take a longer time until reaching large-scale deployment. In the mean time the described scheme can be used for additional TLS server certificate validation, and can be deployed rapidly as it only requires client-side support. It also addresses weaknesses of an earlier approach implemented by Certificate Patrol, resulting in a more reliable TOFU pinning scheme with less unnecessary notifications presented to the user.

We have implemented a proof of concept of the verification, storage, and notification functionality described here. Further work has to be done regarding the integration of the proposed design into applications and libraries.

References

[1] *Comodo Report of Incident*. URL: http://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html.

[2] *Black Tulip, Report of the investigation into the DigiNotar Certificate Authority breach*. Tech. rep. 2012. URL: http://www.rijksoverheid.nl/bestanden/documenten-en-publicaties/rapporten/2012/08/13/black-tulip-update/black-tulip-update.pdf.

[3] *Rogue intermediate CAs issued by TURKTRUST*. URL: http://googleonlinesecurity.blogspot.de/2013/01/enhancing-digital-certificate-security.html.

[4] Christopher Soghoian and Sid Stamm. *Certified lies: Detecting and defeating government interception attacks against SSL*. Tech. rep. 2010.

[5] Werner Koch and Marcus Brinkmann. "STEED – Usable End-to-End Encryption". In: (2011). URL: http://www.g10code.de/docs/steed-usable-e2ee.pdf.

[6] *Key Management through Key Continuity (KCM)*. 2008. URL: https://tools.ietf.org/html/draft-gutmann-keycont-01.

[7] *Certificate Patrol*. 2009. URL: http://patrol.psyced.org/.

[8] Dan Wendlandt, David G. Andersen, and Adrian Perrig. *Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing*. 2008.

[9] *Convergence*. URL: http://convergence.io/.

[10] R. Holz et al. "X. 509 Forensics: Detecting and Localising the SSL/TLS Men-in-the-middle". In: *Computer Security–ESORICS 2012* (2012), pp. 217–234.

[11] Adam Langley. *Why not Convergence?* 2011. URL: http://www.imperialviolet.org/2011/09/07/convergence.html.

[12] P. Hoffman and J. Schlyter. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*. 2012. URL: https://tools.ietf.org/html/rfc6698.

[13] M. Marlinspike and T. Perrin. *Trust Assertions for Certificate Keys*. 2012. URL: http://tack.io/draft.html.

[14] C. Evans, C. Palmer, and R. Sleevi. *Public Key Pinning Extension for HTTP*. 2012. URL: https://tools.ietf.org/html/draft-ietf-websec-key-pinning-04.

[15] *The Monkeysphere Project*. URL: http://web.monkeysphere.info/.

[16] et al. Wouters. *Out-of-Band Public Key Validation for Transport Layer Security (TLS)*. URL: https://tools.ietf.org/html/draft-ietf-tls-oob-pubkey-06.

[17] Adam Langley. *Public key pinning*. 2011. URL: http://www.imperialviolet.org/2011/05/04/pinning.html.

[18] Thomas Roessler and Anil Saldhana. *Web Security Context: User Interface Guidelines*. 2010. URL: http://www.w3.org/TR/wsc-ui/.

[19] *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. 2008. URL: https://tools.ietf.org/html/rfc5280.

[20] Stef Walter. *Sharing Trust Policy aka. Stapled Certificate Extensions*. 2013. URL: http://p11-glue.freedesktop.org/doc/sharing-trust-policy/.

[21] M. Georgiev et al. "The most dangerous code in the world: validating ssl certificates in non-browser software". In: *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM. 2012, pp. 38–49.