Are Certificate Thumbprints Unique?

Greg Zaverucha
Microsoft Research
gregz@microsoft.com

Dan Shumow
Microsoft Research
danshu@microsoft.com

October 1, 2018

Abstract

A *certificate thumbprint* is a hash of a certificate, computed over all certificate data and its signature. Thumbprints are used as unique identifiers for certificates, in applications when making trust decisions, in configuration files, and displayed in interfaces. In this paper we show that thumbprints are not unique in two cases. First, we demonstrate that creating two X.509 certificates with the same thumbprint is possible when the hash function is weak, in particular when chosen-prefix collision attacks are possible. This type of collision attack is now practical for MD5, and expected to be practical for SHA-1 in the near future. Second, we show that certificates may be mauled in a way that they remain valid, but that they have different thumbprints.

While these properties may be unexpected, we believe the scenarios where this could lead to a practical attack are limited and require very sophisticated attackers. We also checked the thumbprints of a large dataset of certificates used on the Internet, and found no evidence that would indicate thumbprints of certificates in use today are not unique.

1 Introduction

A *certificate thumbprint*, also called a fingerprint, is a hash of a certificate, computed over all certificate data and its signature. Thumbprints are used as unique identifiers for certificates, in applications when making trust decisions, in configuration files, and displayed in interfaces. Due to the variety of uses for thumbprints, it is not immediately clear what, if any, their security needs are. Thumbprints are usually implemented with cryptographic hash functions and used in security applications, in some cases as an implementation technique and not as a core security mechanism. This use is not unlike the hash function used in a hash table; innocuous details alongside more security-critical components. In these cases, migrating thumbprints from weak or broken hash functions may seem like a low priority or unnecessary.

In other cases, when making trust decisions, thumbprints must be unique, e.g., when deciding whether to authorize the subject of a certificate, an attacker with a certificate

having the same thumbprint as an honest user may be incorrectly granted access. In this paper we investigate the security of thumbprints, namely the the hash function security properties required to ensure thumbprints are unique.

We consider two ways that thumbprints can be (non-)unique. First, given a thumbprint, it should uniquely identify a certificate (Property U1), or equivalently, no two certificates should have the same thumbprint. Second, given a certificate, there should be a unique thumbprint associated with it (Property U2). We say that a thumbprint is U1-unique if it satisfies Property U1 (U2-unique is defined similarly).

Property U2 We show that Property U2 does not hold by demonstrating simple ways that thumbprints may fail to uniquely identify a certificate. For certificates signed with ECDSA, the signature value may be modified, yet remain a valid signature on the same message. This allows a certificate holder to modify their certificate, in an otherwise benign way, to change the thumbprint. This could have security implications in systems that use thumbprints to revoke access. We also give a similar modification for RSA-signed certificates, however it works on the encoding of the signature, rather than at the cryptographic level.

Property U1 Identical thumbprints for different certificates (Property U1) is a collision for the hash function used to compute the thumbprint, so clearly a collision-resistant hash function is sufficient to guarantee Property U1. However, SHA-1 is commonly used for computing certificate thumbprints, and trust decisions are made based on thumbprints. With the recent demonstration that SHA-1 is not collision-resistant, and existing code still relying on SHA-1 thumbprints, we need a more precise answer to determine risk to applications. Collision resistance is sufficient, but is it necessary? If not, what hash function properties are required for thumbprint security?

After a quick look, it might seem that a second pre-image attack on the hash function used to compute the fingerprint breaks U1-uniqueness. Given a first certificate and associated thumbprint, the attacker must find a second certificate, with a chosen public key. Abstractly this is a second preimage problem, but the specific format of the second certificate is significant. We now argue that a second preimage attack alone is not sufficient, even when the CA signing key is known to the attacker.

Consider the case of a CA that issues certificates signed with a signature algorithm that uses a strong hash algorithm like SHA-256 (denoted H_{sign}), and a relying party that computes thumbprints with a weak hash function (denoted H) for which second preimage attacks are possible. The first challenge is that an honest CA issuing the certificate will typically insert unpredictable values such as a serial number. Given that serial numbers are generally 8–20 pseudorandom bytes, guessing the serial number is infeasible.

Now suppose the CA is malicious, so the attacker may use the signing key (the keypair is (pk, sk)) and choose the certificate data (like the serial number). The attacker now faces

the following problem: Given a certificate C_1 and sk, find a second certificate C_2 of the form $P||S$ such that $H(C_1) = H(C_2)$, and S is a valid signature on P with respect to pk. This appears to be hard, at least for standardized signatures algorithms like ECDSA, RSA-PSS and RSA-PKCS#1-v1.5. Intuitively, this is because for certificates to collide, the attacker needs control over the last part of C_2, because of the iterated nature of the Merkle-Damgård construction. But the last part of C_2 is the signature, which depends on $H_{sign}(P)$, and since H_{sign} is a secure hash function, getting a specific signature value (for any message) is difficult, as we discuss in Section 6.2.

The point of this example is to show that the question of SHA-1 thumbprint security does not have an obvious answer, and both the signature algorithm used by the certificate issuer, and the digest algorithm used to compute the thumbprint can impact security.

Previous work by Stevens et al. [33, 35] created colliding TBSCertificate data[1], and then used a signature on legitimate TBSCertificate to authenticate a malicious certificate. The attack used a *chosen-prefix collision attack* on H, meaning the attacker can choose two prefixes P and P', and can then find S and S' such that $P||S$ and $P'||S'$ have the same digest. Note that with Merkle-Damgård functions (like SHA-1, SHA-2 and MD5) if $P||S$ and $P'||S'$ collide, then $P||S||T$ and $P'||S'||T$ also collide.

Stevens et al. [35] demonstrated practical attacks on PKI when CAs use weak hash functions for signing certificates. Since that work (and in part because of it), CAs have largely switched to signature algorithms with strong digests like SHA-256. By contrast, thumbprints are computed by applications controlled by a disparate set of relying parties, and we have limited information about whether they have switched to strong functions. Given that SHA-1 is the default thumbprint digest algorithm in Windows and OpenSSL (and has been for many years), it's likely still in widespread use.

In this paper, we demonstrate how a chosen-prefix collision attack on the thumbprint digest algorithm, combined with a previously known property of RSA and ECDSA signatures is sufficient in practice to create two X.509 certificates with the same thumbprint. We then conclude that resistance to chosen-prefix collisions is necessary for U1-unique thumbprints. For our demo to work, the signature algorithm must be vulnerable to *key substitution attacks*. In a key substitution attack, the attacker is given a signature on a message that verifies under a first public key, and they must find a second key pair such that the signature remains valid under the new public key. For this work, it's important to note that the message used in the second verification can be different. This property is not an attack under the common definition of secure signatures (GMR security [12]), and only impacts security in practice in some less common applications. Perhaps this is why standards for ubiquitous signature algorithms like ECDSA and RSA do not mitigate it. Essentially every X.509 certificate in use today is signed with an algorithm that allows a form of key substitution attack.

Using a key substitution attack, we can create two certificates with the same signature,

[1]TBSCertificate is everything in an X.509 certificate except the signature, we review this in Section 2.3.

both valid under different issuer keys. The chosen-prefix collision attack allows us to have different attributes in the two certificates, yet still have the same thumbprint.

Impact on existing software and systems It is difficult to fully assess the impact of this finding, as many of the systems and applications using thumbprints are proprietary. Further, use of thumbprints is not standardized, so aside from some general patterns, details of their use will be particular to a given system, and require a separate security analysis. Using thumbprints in the context of certificate pinning is one common use, and we examine it in some detail. There are some mitigating factors that make us conclude that this will not put deployed systems at risk in the short term.

- Efficient chosen-prefix collision attacks against SHA-1 have not been demonstrated yet, and MD5 is not commonly used for computing thumbprints.
- Attacks in the scenarios we investigated (e.g., certificate pinning) would be sophisticated, and are likely to be out of reach to non-nation state attackers.
- As part of our investigation, we checked the thumbprints of roughly 125 million X.509 certificates publicly accessible on the Internet. We checked for colliding SHA-1 and MD5 thumbprints (and found none), and also used hash function collision detection (§2.1) to check whether any of these certificates are part of a colliding pair of certificates. Again, none were found. We also checked for evidence of key substitution attacks, and found none. Because of this we have confidence that all certificates in use on the Internet have unique thumbprints.

That said, we believe SHA-1 should be replaced with a stronger hash algorithm, and explain our recommendations in Section 7. We believe most people would expect, though this is not stated anywhere, that CAs should not be able to create multiple certificates with the same thumbprint, and would see this as unexpected and undesirable.

1.1 Paper Summary

We start with a review of some background material in Section 2. In Section 3 we explain how to create two certificates with the same thumbprint using key substitution and chosen prefix collision attacks, we also explain our implementation and give a pair of certificates with the same MD5 thumbprint in Appendix A. Section 4 shows ways that U2-uniqueness does not hold for thumbprints, and we give example certificates that have two thumbprints in Appendix B. Section 5 describes our investigation of X.509 certificates in use on the Internet. Section 6 discusses some interesting approaches to finding colliding thumbprints that were not quite practical. We conclude with some recommendations to improve thumbprint security in Section 7.

2 Background and Related Work

2.1 Hash function security

Let $H : X \to Y$ be a cryptographic hash function. Recall that a collision attack against H is efficiently finding distinct M and M' in X such that $H(M) = H(M')$. A second preimage attack is, given $x \in X$, efficiently find $x' \in X$ such that $x' \neq x$ and $H(x') = H(x)$. If collision attacks are hard, we say that H is collision resistant similarly, H can be second-preimage resistant.

Chosen-prefix collision attacks In a chosen-prefix collision attack, we can choose arbitrary prefix data, P and P', and then find values S and S' such that $P||S$ and $P'||S'$ have the same hash. The values S and S' are outside of attacker control, and are sometimes called "tumor" values, since they appear in final document or certificate, as random, useless data. It's also possible to append an arbitrary value T to both messages, so that $P||S||T$ and $P'||S'||T$ collide, because the Merkle-Damgård construction is iterated.

MD5 Practical collisions for MD5 were first disclosed in 2004 by Wang and Yu [37]. This collision was for random data. Less than three years later, the practical collision attacks became more flexible, when chosen-prefix collisions were made practical by Stevens, Lenstra and de Weger [33]. Once chosen-prefix collisions were possible, and the implications understood, we saw demonstrations of attacks by researchers on public-key infrastructure: the colliding certificates in [33], a rogue CA in [35] and finally a real-world, nation-state attack with the Flame malware [11]. The tumor values for chosen-prefix collision attacks are as short as 76 bytes, however shorter tumors require more computational effort to find (e.g., about 2^{49} MD5 calls for a three-block tumor instead of 2^{16} calls for a nine block tumor).

SHA-1 With SHA-1, there is a long history of steadily improving theoretical attacks (see the references of [32]), with an actual collision being demonstrated in 2017 by Stevens et al. [32]. Chosen-prefix collision attacks have not yet been demonstrated against SHA-1. In 2013 Stevens [31] found a chosen-prefix collision attack against SHA-1 requiring approximately $2^{77.1}$ SHA-1 compression calls, which is over sixteen thousand times more expensive than the $2^{63.1}$ SHA-1 compression calls used to find the identical-prefix collision in [32] [2]. The size of the tumors is roughly 128 bytes (two SHA-1 blocks). Already this attack is arguably practical, for a sufficiently well-funded attacker. We don't try to predict when chosen-prefix collision attacks on SHA-1 will be efficient enough to be demonstrated

[2] In the rump session at Eurocrypt 2018, Leurent and Peyrin gave a presentation with an overview of new techniques for finding SHA-1 chosen-prefix collisions with complexity between $2^{66.9}$ and $2^{69.3}$. At the time of writing, the paper was not available.

by researchers, but feel that it is already too risky to rely on the chosen-prefix collision resistance of SHA-1 for security.

Collision Detection For MD5 and SHA-1 it is possible to efficiently detect inputs that are part of a collision attack [30, 34]. That is, given M such that $H(M) = H(M')$ and M and M' were constructed using known attacks on H, we can recognize that M is part of a colliding pair, without knowing M'. We use this in our analysis of certificates on the Internet, and discuss it further in Section 5.

2.2 Key Substitution Attacks

In a *key substitution attack* (also called a *duplicate signature key selection attack*), the attacker is given a signature on a message that verifies under a first public key, and he must find a second public key such that the signature remains valid under the new public key. In [3, 15] Menezes et al. analyzed common signature algorithms with respect to key substitution attacks. Later, Bohli et al., consider the case when the secret key is known [4]. For colliding certificate thumbprints if the issuer is compromised or malicious, his private key may be used, so this extended model is relevant to our work. If the private key is used, then key substitution attacks are possible for ECDSA signatures. We are also interested in the case where the signature is verified with a different message as well as a different public key. Key substitution attacks are most feasible on RSA but also ECDSA, as we detail below.

RSA signatures We are given messages m_1 and m_2, and an RSA public key (N_1, e_1) and a signature $s = E(m_1)^{d_1} \pmod{N_1}$, where E is an encoding function that maps a message to an integer modulo N_1. For PKCS#1 v1.5, E hashes m_1 and adds padding bytes. For RSA-PSS, E is more complicated, but what we describe here is independent of the choice of E. Choose $N_2 = p_2 q_2$ such that $\phi(N_2)$ is smooth enough that we can use the Pohlig-Hellman algorithm to solve discrete logarithms in in $\mathbb{Z}^*_{N_2}$. Simultaneously, ensure that N_2 is difficult to factor with Pollard's $p-1$ algorithm, by making sure the factors of $\phi(N_2)$ are large enough, e.g., 80 or more bits. We also need to ensure that s and $E(m_2)$ are generators of $\mathbb{Z}^*_{p_2}$ and $\mathbb{Z}^*_{q_2}$. Then we can efficiently find d_2 such that $E(m_2)^{d_2} = s$, or equivalently, $d_2 = \log_{E(m_2)} s \pmod{N_2}$. Once we have d_2, compute $e_2 = {d_2}^{-1} \pmod{\phi(N_2)}$, and output (N_2, e_2, p_2, q_2).

ECDSA To set notation we briefly review ECDSA. Let g generate a group of points on an elliptic curve of prime order n, written multiplicatively. A key pair (pk, sk) is (g^x, x) where $x \in \mathbb{Z}_n$. A signature on m_1 is (r, s) where $r = f(g^k)$, for a random value k, and f takes the x-coordinate of the point g^k. The value s is computed $s = (e_1 + rx)/k \pmod{n}$ where e_1 is an encoding of $H(m_1)$. Verification checks that $r = f(z)$ where $z = g^{e_1/s} pk^{r/s}$. In [3] a key substitution attack is shown when g is part of the public key, different for each

user. Since most standards fix g as part of the domain parameters, this will not work in practice.

However, if the secret key x is known, then we can compute a second key pair (pk_2, x_2) such that (r, s) is a valid signature of a new message m_2 under (pk_2, x_2). Let e_2 be the encoding of $H(m_2)$, then note that verification would compute $z = g^{e_2/s} {pk_2}^{r/s}$. It can be shown that verification will succeed for the choice $x_2 = (e_1 + rx - e_2)/r$ and $pk_2 = g^{x_2}$.

In another variant that does not require knowledge of x, the second public key is computed as $pk_2 = R'^{s/r} g^{-e_2/r}$, where R' is a point with x-coordinate equal to r, and the corresponding secret key x_2 is unknown [4, Remark 4]. Verification of m_2 with public key pk_2 and signature (r, s) will succeed, but no private key operations for pk_2 can be done.

EdDSA is a new elliptic curve signature algorithm, not currently used to sign certificates, but is not vulnerable to key substitution attacks, and does not have malleable signatures.

EdDSA EdDSA is also an elliptic curve scheme, so we re-use the notation from ECDSA. A signature is a pair (R, s) where R is a point on the curve, computed as g^k for a random value k, and $s = k + cx \pmod n$, where $c = H(R||pk||m)$. Verification re-computes $c = H(R||pk||m)$ and checks whether $g^s = R \cdot pk^c$. Even with knowledge of x, a key substitution attack is not possible, since the signer's public key is prepended to the message being signed (see the analysis and discussion in [19, 15]).

The overwhelming majority of the 126M certificates in our dataset (§5) were signed with RSA PKCS#1 (99.6%), then there were about 236K ECDSA-signed certificates and 204K certificates signed with RSA-PSS.

2.3 Review of X.509 Certificates

In this section we review X.509 certificates, in sufficient detail to explain our process for creating certificates with colliding thumbprints. The main components of a certificate are the subject's name and public key, the issuer's name, a serial number, validity period and a signature authenticating these attributes.

X.509 certificate data is encoded according to the following ASN.1 structure from RFC 5280 [8].

```
Certificate  ::=  SEQUENCE  {
     tbsCertificate       TBSCertificate,
     signatureAlgorithm   AlgorithmIdentifier,
     signature            BIT STRING  }

TBSCertificate  ::=  SEQUENCE  {
     version         [0]  Version DEFAULT v1,
     serialNumber         CertificateSerialNumber,
     signature            AlgorithmIdentifier,
     issuer               Name,
```

```
validity             Validity,
subject              Name,
subjectPublicKeyInfo SubjectPublicKeyInfo,
issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
extensions      [3]  Extensions OPTIONAL }
```

The majority of certificates in use are version 3 (about 83% in our dataset), where the extensions field must be understood by relying parties (RP). Extensions of X.509 version 3 certificates can include critical additional attributes, such as key usage, or whether the certificate is a CA certificate or an end-entity (EE) certificate.

Certificates are commonly encoded as DER, a binary representation of the ASN.1 `Certificate` structure, or PEM, an ASCII format that contains the base64 encoding of the DER. More description of these encodings can be found in [8].

When a subject requests a certificate from a *certificate authority* (CA), they send a standardized message called a *certificate signing request* (CSR) that contains their name and public key, along with optional additional information [22]. The CSR is signed with the subject's public key, demonstrating possession of the corresponding private key.

2.4 How thumbprints are computed

When computing the thumbprint of a certificate, the entire Certificate ASN.1 structure is input to a hash function, like SHA-256. The ASN.1 structure is encoded with the DER encoding (which is unambiguous). There is not a standard specifying how to compute thumbprints, but the Windows shell, .NET [1], and OpenSSL all use this de facto standard, with SHA-1 as the default hash algorithm.[3] The lack of a standard is understandable, since thumbprints are typically local to an application or system, and interoperability between systems is not required.

3 Creating Two Certificates with the Same Thumbprint

We describe how to create a pair of certificates with the same thumbprint, given a chosen-prefix collision attack on H and a key substitution attack on the signature algorithm used to sign the certificates. This demonstrates that certificates are not U1-unique when the hash function is weak.

See Figure 1 for an overview of the demo. In the demo, a malicious or compromised CA, denoted CA*, issues a certificate to an honest subject, and creates a second certificate with the same thumbprint. The first certificate, C_1, is issued by Issuer1 (the CA), and is issued to the honest subject, for which the private key is unknown to CA*.

[3]For example, to compute a thumbprint of a PEM encoded certificate with openssl, use the command `openssl x509 -fingerprint -in certificate.crt`.

The second certificate, C_2, is issued by Issuer2. The CA creates Issuer2 as a new (possible intermediate) CA, with a chosen key pair. The subject key pair in C_2 is chosen by CA*. Issuer2's key pair is chosen such that the signature of C_1 is also a valid signature of C_2 under Issuer2's key. Choosing Issuer2's key is done with a key substitution attack, as described in Section 2.2.

Certificate C_1 has the public key and name from the CSR. It can be signed with an existing intermediate CA, for which CA* does not know the private key. C_1 is crafted to have the form $P||S||T$, where

P is the beginning of `tbsCertificate`, including all but the last extension, the *subject key identifier* (SKI) [8, §4.2.1.2]. This includes the version, serial number, issuer, and the subject public key.

S is the SKI, set to the tumor value. This SKI will be longer than usual, and not computed as recommended by [8], but it is allowed by the standard, and the software we tried ignores long SKI values in end-entity certificates (where it arguably doesn't really have a purpose, since EE certificates do not issue further certificates). The SKI is one of the X.509 v3 extensions, and since extensions may be encoded in any order, we can choose to put it last, giving CA* control to choose arbitrary values for the extensions in C_2 (e.g., some other interesting extensions are (extended) key usage, basic constraints, and CRL distribution points).

T is the signature of $P||S$, created with Issuer1's signing key in the normal way.

Certificate C_2 is constructed to have the same thumbprint as C_1. It is issued by a new intermediate CA, denoted Issuer2, and Issuer2's public key is created after seeing the CSR. C_2 is of the form $P'||S'||T$, where

P' has the same format as P but the attributes can differ arbitrarily. P' has a different serial number, issuer, and the subject public key is chosen by CA*.

S' is similar to S, an SKI attribute set to the second tumor value.

T is the signature, identical to T in C_1.

Trusting Issuer2 In addition to choosing Issuer2's key pair at the time of the attack, the attacker may also require that Issuer2 is a valid intermediate CA of a root trusted by the relying party, or be a root itself. It depends on how the RP has implemented validation (and optionally pinning). If the RP looks only at the leaf cert, Issuer2 does not need to be trusted. If EE certs are third in the chain (or further), the same issuer that created Issuer1 may issue a CA certificate to Issuer2, and Issuer2 will be immediately trusted by RPs, without an update to the root store.

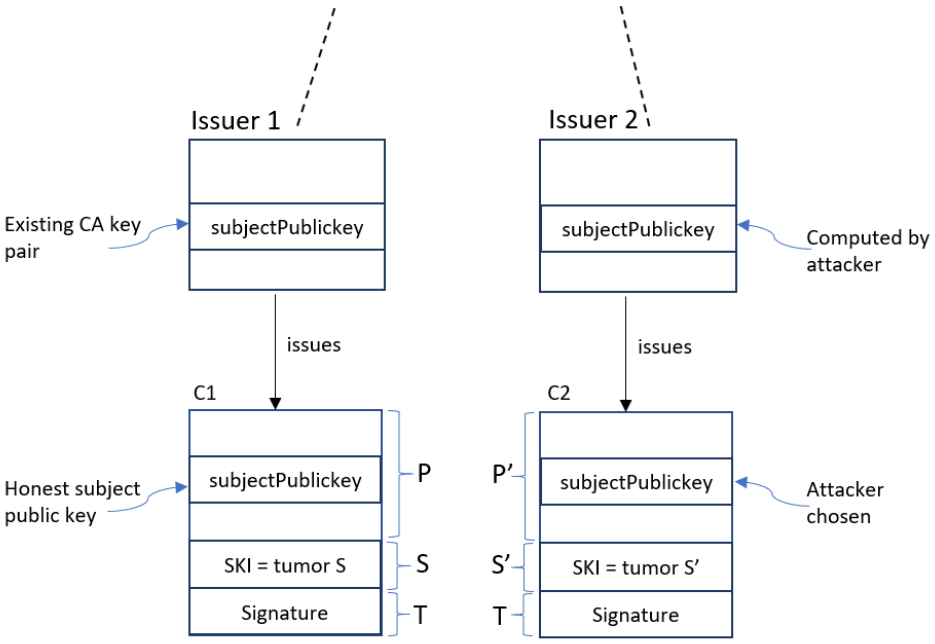

Figure 1: Scenario used to create certificates C_1 and C_2, with colliding thumbprints.

Pinning to C_1 Note that if a relying party pins C_1, then even the CA that issued C_1 cannot masquerade as the subject of C_1 (Subject1). For example, without pinning, Issuer1 could issue additional certificates for Subject1 certifying keys it has generated, and these certificates would be trusted by the RP. But if the RP has pinned C_1, other certificates issued by Issuer1 will not be trusted. Therefore, the thumbprint collision described here could allow a CA to intercept traffic between Subject1 and the RP, or masquerade as Subject1, in a way not possible when thumbprints are unique.

3.1 Detailed Steps

We now describe the steps above in more detail. To make things concrete we assume the signing algorithm used by Issuer1 is RSA (with either padding mode from PKCS#1 [20]).

1. The honest party submits a CSR for C_1. The subject name is Subject1.
2. CA* computes tbsCertificate1, the `TBSCertificate` data for C_1, excluding the last field, the SKI.
3. CA* computes an arbitrary RSA keypair, the subjectPublicKey for C_2.
4. CA* computes tbsCertificate2, the `TBSCertificate` data for C_2, excluding the last

field, SKI. Note that the issuer is Issuer2, but Issuer2's public key is not present in the certificate.

5. Using the chosen-prefix collision attack, CA* computes the tumor values S and S', for the prefix values tbsCertificate1 and tbsCertificate2, then completes tbsCertificate1 and tbsCertificate2 by setting the SKIs to S and S', respectively. Now tbsCertificate1 and tbsCertificate2 collide under H (but not H_{sign}).

6. CA* signs tbsCertificate1 with Issuer1's key and completes C_1 and responds to the CSR with it. Denote the signature value σ.

The following steps can be done offline, i.e., at anytime following the creation of C_1.

7. CA* performs the key substitution attack to create an RSA key pair such that σ is a valid signature of tbsCertificate2 with respect to the new public key.

8. CA* forms C_2 as tbsCertificate2 followed by σ.

9. CA* creates a certificate certifying Issuer2, C_{I2}.

10. CA* uses C_2 and C_{I2} to masquerade as Subject1 to clients that have identify Subject1 by the thumbprint of C_1, other for some other malicious purpose.

Notes on Issuer2's RSA key pair

- It will not have the usual public exponent $e = 2^{16} + 1$, but instead e will be a large random value. From our experiments, common software supports large values of e.[4] About 2.2M of the RSA certificates in our dataset (1.9%) use an e value that is not $2^{16} + 1$. These are almost all small values, the largest being 40 bits.

- The RSA modulus N_2 will not be a product of safe primes (i.e., product of p and q such that $(p-1)/2$ and $(q-1)/2$ are also prime).

- It will be infeasible to factor N_2 (provided a sufficient amount of time is spent on the key substitution attack).

- It should be difficult to efficiently distinguish N_2 from an honestly generated modulus.

[4]Though not universally, for example, for moduli larger than 3072 bits, OpenSSL requires e be at most 64 bits.

Comments

- The SKI is long and random looking on both certs, and some implementations may notice this. But the SKI should be ignored in EE certs, as not all of them have it. Alternatively, we could include the tumor in the (now deprecated) Netscape Comment field as in [35]. We could also use a custom X.509 v3 extension marked non-critical, which should be ignored by implementations.

- The signatures will be the same, and this should never happen in normal circumstances. See our discussion of signature malleability in Section 4 for a ways an attacker can make the signatures different.

- A new intermediate CA, Issuer2, is required for each pair of certs with colliding thumbprints. Since Issuer2's key is secure, it can be used for other purposes to avoid suspicion.

- The key substitution step is easier for the attacker when the CA signature algorithm is RSA. If it's ECDSA the attacker must know Issuer1's signing key, oracle access is insufficient.

3.2 Implementation Details

In order to demonstrate the feasibility of using our method to construct two certificates with the same thumbprint we implemented it, creating two certificates with the same MD5 thumbprint. The issuer keys are RSA-2048, and the signature algorithm is RSA PKCS#1 v1.5 with SHA-256. The two end-entity certificates, along with the issuer certificates and the second issuer's key are given in Appendix A.

Our demo uses the OpenSSL command line tools where possible, mainly for the initial step of creating Issuer1, initial versions of C_1 and C_2 to get the prefixes.

Once we have the prefixes, we used the HashClash [29] software package to find chosen-prefix collisions with MD5. Our prefixes were 488 bytes long, the default settings of HashClash produce tumor values of nine 64-byte MD5 blocks (576 bytes) after rounding 488 to the next multiple of 64, for a total of 600 bytes of overhead. Finding the collision took about four hours on a 32 core server[5]. As shown by Stevens et al. [35], with more computational effort the overhead can be reduced to as little as 76 bytes. As our implementation serves only to demonstrate feasibility, we chose not to spend further resources to reduce the overhead of the chosen-prefix collision.

For the key substitution attack, we need to extract the tbsCertificate data of C_2, and encode it with PKCS#1 v1.5 encoding, and convert it to an integer. We also needed the signature value from C_1 represented as an integer. For this step we used the mbedTLS library [2].

[5]Specifically, we used an Azure D32s v3 instance.

We then implemented the key substitution step of creating a second modulus (of the correct form) and finding the correct public exponent by solving a discrete logarithm using our implementation of the Pohlig-Hellman algorithm in PARI-GP. We used 2048-bit modulii for all keys in the demo, and for Issuer2's key, the factors of $\phi(N_2)$ were 32 bits. With our implementation this takes about a minute on single core of an Intel Xeon E5-1630 clocked at 3.7 GHz. In practice the attacker would prefer larger factors of $\phi(N_2)$ so that N_2 cannot be factored with Pollard's $p-1$ algorithm. Using 80-bit factors would make the $p-1$ algorithm impractical, while still allowing the discrete logarithm step of the key substitution attack to remain practical (requiring roughly 2^{40} operations). Again, we chose not to optimize this step in our proof of concept, and this step was already shown to be very efficient with modest compute resources by Henry and Goldberg [14]. Once we find Issuer2's key pair as integers, we again use mbedTLS to encode these as ASN.1/PEM, for use with OpenSSL and other programs.

3.3 Generalizing to Other Primitives

We've demonstrated the attack with RSA signatures and MD5 thumbprints. For other hash functions where chosen-prefix collision attacks are practical, we expect our demo to generalize in a straightforward way, since nothing about our work is specific to a particular hash function. The CPU cost and the size of the colliding certificates will depend on hash function.

When different signature algorithms are used, the question is whether a key substitution attack is feasible. We found in our survey (§5) that all certificates in use on the Internet today are signed with an algorithm that allows a strong enough key substitution attack. For ECDSA the attack may require access to the CA's private key, as discussed in Section 2.2.

4 Creating Two Thumbprints for the Same Certificate

Now to Property U2, that a given certificate should have a unique thumbprint. Thumbprints are not U2-unique if the CA signature on the certificate is malleable. By malleable, we mean that given a valid signature of m, we can construct a second, different signature on m. This does not violate the GMR security definition for signatures, since the signer did intend to sign m.

In this case a subject may change the signature in their certificate in order to change the thumbprint. This could have security implications, for example, if a system implemented revocation or banning by thumbprint (instead of by serial number and issuer public key, as is done in OCSP [28]). Our quick search did not find examples in practice where this is an issue.

Cryptographic Malleability Given an ECDSA signature (r, s) on message m, the value $(r, -s \pmod n)$ is also a valid signature on m. A similar malleability is not present in RSA

signatures. In Appendix B we give a pair of ECDSA certificates that were mauled in this way.

Mauling Certificate Signature Encoding Recall that the Certificate ASN.1 structure has three fields: tbsCertificate, signatureAlgorithm, and signatureValue. The signatureAlgorithm is of type AlgorithmIdentifer, defined as follows:

```
AlgorithmIdentifier ::=  SEQUENCE  {
    algorithm          OBJECT IDENTIFIER,
    parameters         ANY DEFINED BY algorithm OPTIONAL  }
```

The optional parameters field is not used for RSA. Also, ASN.1 has a NULL type, indicated with the bytes `05 00`. Thus every certificate can encode the algorithmIdentifier in two valid, equivalent ways, one with a NULL item in the sequence, and one without. The X.509 standard [8, §4.1.2.3] requires that the signatureAlgorithm field be equal to the signature field of the tbsCertificate (which is also of type AlgorithmIdentifier). OpenSSL enforces this, but Windows does not, therefore we can change the DER encoding of the certificate in order to change the thumbprint, but the signature remains valid, since the signatureValue and tbsCertificate fields are unchanged.

We found certificates with this difference in the wild, and include an example in Appendix B. It appears more common to include the NULL item, (e.g., OpenSSL includes it), but we speculate that some software removes it, perhaps as an optimization, or perhaps when serializing and deserializing a certificate with the NULL item.

5 Thumbprints of Certificates on the Internet

Since certificates are public, and often accessible to anyone on the Internet, we were able to compute the thumbprints of many certificates, and check whether they are unique. Project Sonar [27] from Rapid7 uses ZMap [9] to regularly scan the Internet and publishes data sets containing certificates in cooperation with the University of Michigan at `scans.io` [5]. We downloaded all available certificate data from scans before January 20th, 2018, from the "SSL Certificates" [26] and the "More SSL Certificates" [25] data sets, which go back to late 2013. The first set includes certificates from SSL services on TCP port 443 (usually HTTPS), and the second includes certificates from other services that use SSL like IMAP, POP3 and SMTP, sometimes using STARTTLS. This was about 65GB of gzip compressed data, which resulted in 125.8 million unique certificates, i.e., 125.8 million of the certificates had unique SHA-256 thumbprints. The large amount of redundant data is because most certificates appear in multiple scans. We did the following analysis on this data set.

Compute thumbprints We computed the SHA-256, SHA-1, MD5 and MD4 thumbprints of the certificates. We then checked whether any thumbprints repeated for the digests we computed. All thumbprints were unique.

Run collision detection We used a collision detection algorithm for SHA-1 and MD5, when computing the thumbprints. Collision detection [30, 34] gives us a way to check whether any of the certificates in our data set were crafted to have the same thumbprint as another certificate not present in our data set. Intuitively, all known collision attacks against SHA-1 and MD5 require that colliding input messages individually satisfy certain conditions, allowing us to recognize one message of a colliding message pair. For thumbprints, this means we can detect the case when a first certificate has been created as part of an attack, but the second certificate has not been disclosed yet. Thus we can say with high confidence that all certificates in the data set have unique SHA-1 and MD5 thumbprints, among the set of all certificates ever created. Put another way, both certificates with the same thumbprint must be outside of our data set. In theory it's possible to circumvent collision detection with a novel attack, however, since 20+ years of public cryptanalysis on MD5 and SHA-1 did not find such an attack, it would be surprising.

Based on this analysis, we have confidence that for the moment, MD5 and SHA-1 thumbprints of certificates in use on the Internet are U1-unique.

Search for Key Substitution Attacks In order to check for evidence of key substitution attacks, we looked for distinct certificates with the same signature. We found many certificates that shared a signature value with another certificate, but do not believe this is because of a key substitution attack, since they did not appear to make intentional changes to the attributes of the certificate.

There were 2451 certificates that did not have a distinct signature. These formed 701 groups of certificates with the same signature. Of these groups, 598 had size two, 95 had size between 3 and 49, and 8 had size between 50 and 86. The majority of these certificates (64%) were created by devices or deployments where certificates are not verified, usually self-signed certificates, where X.509 is used as a data format to transfer for public keys. In these certificates the public key and signature are fixed, but attributes in the certificates (like an IP address in subjectAltName) frequently change. There is also an SSL proxy that re-writes certain certificates (those it does not trust) without updating the signature, and our dataset contains many instances of the original certificate and the re-written certificate. Some of these certificates have a single byte signature, making it clear that they are just a convenient way to encode the public key.

Another 32% of certificates with non-unique signatures appear to be due to corruption. Network devices store a certificate on disk, it gets corrupted over time, and these corrupted certificates appear as distinct in our dataset. In many instances there are multiple certificates with a single byte difference, suggesting the same fault in storage repeats over time.

The remaining four percent includes 40 certificates with the signature encoding difference explained in Section 4, and certificates that have modifications we did not deem suspicious, possibly made by developers when testing.

All ECDSA signature values (r, s) were unique, but we also checked whether ECDSA

signatures had unique r and s values, in order to detect the signature modification we described in Section 4, where (r, s) is replaced with $(r, -s \pmod n)$. We found 380 certificates that shared an r value with another certificate, and had unrelated s values. These were all self-signed certificates from different instances of the same network device. We speculate this device has a poorly seeded random number generator. We have notified the vendor.

6 Other Attempts to Create Colliding Thumbprints

In this section we discuss two alternative approaches to creating two certificates with the same thumbprint (non U1-unique), one practical one theoretical. Neither succeeds against libraries in use, but the reasons why may be interesting.

6.1 Trailing Data

We investigated the simple approach of appending the tumor value to certificates. For example, take two arbitrary certificates P and P', then use a chosen-prefix collision attack to to compute suffixes S and S' such that $H(P||S) = H(P'||S')$. Depending on how the thumbprint is computed, this may cause a problem. If data $P||S$ is stored in a file, say `cert.der`, then an application that computes thumbprints by hashing the whole file will also hash the suffix and get the colliding thumbprint value. If the thumbprint is computed from the file with OpenSSL[6] or Windows, the code will read the length of the ASN.1 Certificate structure and only hash this many bytes, skipping the suffix. However, hashing the whole file is correct when there is no trailing data, so an application that did so would usually work as expected.

We also tried updating the length of the Certificate structure, in hopes that the thumbprint would be computed over the entire data, and that the ASN.1 parser might ignore the unknown data following the Signature field. The implementations we checked simply fail to parse the certificate in this case.

The final value in the Certificate object is the Signature. We also tried using a larger Signature field, with the tumor appended to the actual signature, to pass length checks done by the ASN.1 parser. Here we hoped that the crypto library would read the first part of the signature buffer (a valid signature), use it, and ignore the suffix. Standards for RSA and ECDSA do enforce a length check on signatures, and indeed we found that OpenSSL, Windows, Java and mbedTLS report the signature as invalid. Interestingly, the mbedTLS RSA implementation verifies the signature successfully first, then checks the length and returns an error. Inspection of the commit history shows that the length check was added after the rest of the implementation[7], so earlier versions would have accepted the certificate

[6] E.g., with the command `openssl x509 -fingerprint -in cert.der -inform DER -md5`

[7] In 2013, by commit id `ac4cd362`.

as valid. This is understandable since testing that primitives fail for many types of invalid inputs is often overlooked. For example, the example RSA tests in the NIST Cryptographic Algorithm Validation Program do not not test for this error [21].

6.2 Chosen Signature Suffix

Continuing with the approach of the previous section where additional data is appended to the signature, we can also ask whether it's possible to generate valid signatures, where the last part of the signature is a chosen value. If this were possible, we might create two certificates with signatures of the form $P||S$ and $P'||S'$ where S and S' are the tumor values produced by chosen-prefix collision attack, that cause the certificate thumbprints to collide.

In this situation, security of thumbprints relies on a nonstandard property of signatures, namely that signers cannot arbitrarily choose the suffix of the signature value. Probabilistic signature schemes always allow signers to choose a small number of trailing bytes arbitrarily with some work; simply generate signatures until the desired suffix occurs by random chance. The length of a suffix required for a chosen-prefix collision attack will probably always be large enough to be out of reach, for example the smallest known suffix for MD5 is 608 bits.

Let *Sign* be a secure signature scheme, and consider the signature scheme $\mathit{Sign}'(sk, M) = \mathit{Sign}(sk, M)||r$, where r is an arbitrary value of fixed length, ignored during verification. This is clearly weakly unforgeable under the common definition if *Sign* is weakly unforgeable. Strong unforgeability would require r to be non-malleable, which seems difficult to achieve if it is a tumor value from a chosen-prefix collision attack that must depend on all data preceding it. Still, Sign' demonstrates that it is possible to have a secure signature scheme with signer-chosen suffixes.

One previously studied property of some signature schemes is called *uniqueness* [18] (or *invariance* [13]), which requires that for every message there exists only a single valid signature value. This is a stronger property, but it implies that signers cannot generate signatures with a chosen suffix. Unfortunately, we did not find an analysis of standardized signature schemes with respect to uniqueness.

We give some informal arguments for why finding signatures with a chosen suffix is difficult for commonly used signature algorithms.

RSA PKCS#1 v1.5 Since the secret exponent d and modulus N are fixed, the problem is to find a message M, such that $(\text{padding}||H(M))^d \mod N$ has the desired suffix. We assume here that there is some flexibility in the choice of M, e.g., it contains a serial number that can be freely chosen. Since exponentiation by $d \pmod N$ is a permutation, if $H(M)$ and $H(M')$ are distinct, then so are their signatures. So there are no choices of d that bias $H(M)^d$ towards the suffix value. When H is a random oracle, $H(M)$ is unpredictable, and the distribution of bit patterns are uniformly random. In the simplest case, when $d = 1$,

finding an M such that $H(M)$ has the desired suffix S requires 2^{ℓ_S} work, where ℓ_S is the bit length of S. Other choices of d do not reduce the search space, so the same amount of work is required.

RSA PSS Similar reasoning applies to PSS, since the encoded message is the output of a strong hash function, finding a message with a chosen suffix appears difficult.

ECDSA and EdDSA First note that for ECC signatures this is usually going to be impossible because of the signatures are too short. For example, ECDSA with curve NIST P256 has signatures of length about 64 bytes, and the smallest tumor for MD5 is 76 bytes. With both ECDSA and EdDSA, the signing equation depends on the output of a strong hash function, in a more complex way than for RSA signatures. It seems difficult to compute signatures with a chosen suffix (but we don't have a proof).

7 Recommendations

To mitigate security issues caused by non-unique thumbprints we recommend the following:

1. Applications should migrate all thumbprints to SHA-256 or SHA-512. If existing infrastructure cannot accommodate a 32-byte digest, truncating these stronger digests to 20 or 16 bytes is preferable to using weaker algorithms.

2. In places where changing the digest is not possible (e.g., out-of-support apps that will not receive an update), or will take time, use collision detection (for MD5 or SHA-1 as appropriate). The collision detection check can be done by a passive network observer for some protocols, such as TLS 1.2 where the client and server certificates are sent unencrypted. An alternative to checking on the network is for the platform to do it as part of certificate validation.

3. Projects that monitor PKI, like Certificate Transparency, should check certificate thumbprints with SHA-1 and MD5 collision detection.

4. Do not use thumbprints to identify and block or revoke certificates in an application, as certificate thumbprints may be changed regardless of the hash function used to compute them. This could be addressed cryptographically by requiring CA signature algorithms be strongly unforgeable, however, this would prevent the use of ECDSA, which is is already in use and supported in most software, so we don't see forbidding it as a practical recommendation.

Non-recommendations The following partial mitigations are not recommended, since they do not address the root cause.

1. Rejecting RSA keys with $e \neq 2^{16} + 1$. This only works for RSA, not ECDSA. It may also be impractical to reject these certificates, since our dataset includes a large number of RSA certificates with other values of e.

2. Rejecting certificates with long SKI values, or custom extensions, or otherwise profiling the X509 standard (RFC 5280). RPs should certainly not do this since the extension may be legitimate and they simply don't understand it (though others do). It may make sense for subjects to scrutinize certificates they are issued, checking for unexpected extensions, or running collision detection on certificates issued to them.

3. We also considered mitigations to prevent key substitution attacks, such as modifying the issuer's signing algorithm by prepending the issuer's public key to the tbsCertificate data. We encourage this for new algorithms (as was done for EdDSA), but think changing existing algorithms is too disruptive, relative to our recommendations above.

References

[1] .NET framework 4.7.1 documentation, X509Certificate2.Thumbprint Property.

[2] ARM Limited. SSL Library mbedTLS/PolarSSL. `https://tls.mbed.org`.

[3] Blake-Wilson, S., and Menezes, A. Unknown key-share attacks on the station-to-station (STS) protocol. In *PKC'99* (Mar. 1999), H. Imai and Y. Zheng, Eds., vol. 1560 of *LNCS*, Springer, Heidelberg, pp. 154–170.

[4] Bohli, J.-M., Rohrich, S., and Steinwandt, R. Key substitution attacks revisited: Taking into account malicious signers. International Journal of Information Security **5**, 2006.

[5] Censys Team at the University of Michigan. Internet-wide scan data repository. `https://scans.io/`. Accessed February 2018.

[6] Chothia, T., Garcia, F. D., Heppel, C., and Stone, C. M. Why banker bob (still) can't get TLS right: A security analysis of TLS in leading UK banking apps. In *FC 2017* (Apr. 2017), A. Kiayias, Ed., vol. 10322 of *LNCS*, Springer, Heidelberg, pp. 579–597.

[7] Clark, J., and van Oorschot, P. C. SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. In *2013 IEEE Symposium on Security and Privacy* (May 2013), IEEE Computer Society Press, pp. 511–525.

[8] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and Polk, W. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. RFC 5280, May 2008. `https://tools.ietf.org/html/rfc5280`.

[9] Durumeric, Z., Wustrow, E., and Halderman, J. A. Zmap: Fast internet-wide scanning and its security applications. 47–53.

[10] Fahl, S., Harbach, M., Perl, H., Koetter, M., and Smith, M. Rethinking SSL development in an appified world. In *ACM CCS 13* (Nov. 2013), A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds., ACM Press, pp. 49–60.

[11] Fillinger, M., and Stevens, M. Reverse-engineering of the cryptanalytic attack used in the Flame super-malware. In *ASIACRYPT 2015, Part II* (Nov. / Dec. 2015), T. Iwata and J. H. Cheon, Eds., vol. 9453 of *LNCS*, Springer, Heidelberg, pp. 586–611.

[12] Goldwasser, S., Micali, S., and Rivest, R. L. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing 17*, 2 (Apr. 1988), 281–308.

[13] Goldwasser, S., and Ostrovsky, R. Invariant signatures and non-interactive zero-knowledge proofs are equivalent (extended abstract). In *CRYPTO'92* (Aug. 1993), E. F. Brickell, Ed., vol. 740 of *LNCS*, Springer, Heidelberg, pp. 228–245.

[14] Henry, R., and Goldberg, I. Solving discrete logarithms in smooth-order groups with CUDA. In Workshop Record of SHARCS, 2012. Available online CACR Tech Report 2012-02, `http://cacr.uwaterloo.ca/techreports/2012/cacr2012-02.pdf`.

[15] Koblitz, N., and Menezes, A. Another look at security definitions. *Advances in Mathematics of Communications*, 7 (2013), 1–38.

[16] Kranch, M., and Bonneau, J. Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning. In *NDSS 2015* (Feb. 2015), The Internet Society.

[17] Laurie, B., Langley, A., and Kasper, E. Certificate transparency. RFC 6962, 2013. `https://tools.ietf.org/html/rfc6962`.

[18] Lysyanskaya, A. Unique signatures and verifiable random functions from the DH-DDH separation. In *CRYPTO 2002* (Aug. 2002), M. Yung, Ed., vol. 2442 of *LNCS*, Springer, Heidelberg, pp. 597–612.

[19] Menezes, A., and Smart, N. Security of signature schemes in a multi-user setting. *Designs, Codes and Cryptography 33* (2004), 261–274.

[20] Moriarty, K., Kaliski, B., Jonsson, J., and Rusch, A. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017, 2016. `https://www.rfc-editor.org/info/rfc8017`.

[21] National Institute for Standards and Technology. Cryptographic algorithm validation program, CAVP testing: Digital signatures. `https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/digital-signatures`. Accessed February 2018.

[22] Nystrom, M., and Kaliski, B. PKCS# 10: Certification request syntax specification version 1.7. RFC 2986, 2000. `https://www.rfc-editor.org/rfc/rfc2986.txt`.

[23] Oltrogge, M., Acar, Y., Dechand, S., Smith, M., and Fahl, S. To pin or not to pin—helping app developers bullet proof their tls connections. In *USENIX Security Symposium* (2015), pp. 239–254.

[24] OWASP (Open Web Application Security Project). Certificate and public key pinning. `https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning`. Accessed February 2018.

[25] Rapid7. Internet-wide scan data repository, more SSL certificates (non-443). `https://scans.io/study/sonar.moressl`.

[26] Rapid7. Internet-wide scan data repository, SSL certificates. `https://scans.io/study/sonar.ssl`.

[27] Rapid7. Project sonar. `https://sonar.labs.rapid7.com/`. Accessed February 2018.

[28] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and Adams, C. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP. RFC 6960, 2013. `https://www.rfc-editor.org/info/rfc6960`.

[29] Stevens, M. Project HashClash. `https://marc-stevens.nl/p/hashclash/`. Code at `https://github.com/cr-marcstevens/hashclash`.

[30] Stevens, M. Counter-cryptanalysis. In *CRYPTO 2013, Part I* (Aug. 2013), R. Canetti and J. A. Garay, Eds., vol. 8042 of *LNCS*, Springer, Heidelberg, pp. 129–146.

[31] Stevens, M. New collision attacks on SHA-1 based on optimal joint local-collision analysis. In *EUROCRYPT 2013* (May 2013), T. Johansson and P. Q. Nguyen, Eds., vol. 7881 of *LNCS*, Springer, Heidelberg, pp. 245–261.

[32] Stevens, M., Bursztein, E., Karpman, P., Albertini, A., and Markov, Y. The first collision for full SHA-1. In *CRYPTO 2017, Part I* (Aug. 2017), J. Katz and H. Shacham, Eds., vol. 10401 of *LNCS*, Springer, Heidelberg, pp. 570–596.

[33] Stevens, M., Lenstra, A. K., and de Weger, B. Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In *EUROCRYPT 2007* (May 2007), M. Naor, Ed., vol. 4515 of *LNCS*, Springer, Heidelberg, pp. 1–22.

[34] Stevens, M., and Shumow, D. Speeding up detection of SHA-1 collision attacks using unavoidable attack conditions. In *26th USENIX Security Symposium (USENIX Security 17)* (Vancouver, BC, 2017), USENIX Association, pp. 881–897.

[35] Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A. K., Molnar, D., Osvik, D. A., and de Weger, B. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In *CRYPTO 2009* (Aug. 2009), S. Halevi, Ed., vol. 5677 of *LNCS*, Springer, Heidelberg, pp. 55–69.

[36] Stone, C. M., Chothia, T., and Garcia, F. D. Spinner: Semi-automatic detection of pinning without hostname verification. In *Proceedings of the 33rd Annual Computer Security Applications Conference* (New York, NY, USA, 2017), ACSAC 2017, ACM, pp. 176–188.

[37] Wang, X., and Yu, H. How to break MD5 and other hash functions. In *EUROCRYPT 2005* (May 2005), R. Cramer, Ed., vol. 3494 of *LNCS*, Springer, Heidelberg, pp. 19–35.

A Example Certificates with Colliding MD5 Thumbprints

Below are four PEM encoded certificates, `cert1.pem`, issued by `issuer1.pem`, and `cert2.pem` issued by `issuer2.pem`. We also include `issuer2.key`, the second issuer's private key, that we solved for with the key substitution attack.

To compute the MD5 thumbprint of `cert1.pem`, use

```
openssl x509 -in cert1.pem -outform DER | openssl dgst -md5
```

To check that `issuer1.pem` is the issuer of `cert1.pem`, use

```
openssl verify -CAfile issuer1.pem -verbose \;
    -attime 1519522786 cert1.pem
```

`cert1.pem`

```
-----BEGIN CERTIFICATE-----
MIIFUDCCBDigAwIBAgIRAP8ttrDYI6IBq/y6yjjBzZQwDQYJKoZIhvcNAQELBQAw
EjEQMA4GA1UEAwwHRGVtb0NBMTAiGA8yMDE4MDIxNDEyMDAwMFoYDzIwMTkwMjE0
MTIwMDAwWjAyMRowGAYDVQQLDBFEZW1vIENlcnRpZmljYXRlMTEUMBIGA1UEAwwL
ZXhhbXBsZS5vcmcwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCnWZHc
58rhUX35F84trHHUDlLLCcYidPdVK1XFAiUvypoAI4gb2wux/+1i30cQWGR8eKIq
MY+Rbrke0y967nPLcZ23Q94iGIIGTNyCKqGnzTEoe0mLu+CDAT0q/lFjnYawW7di
7/rlqBlz+9b0Q9J3if7siYf3fMOsw890ExWi50RgMGKkT5DIdaOHu/COhwOae/Bj
fQdiafgSRPcb8ldr3EIy216Z3GyEZ6dpvpCggmeRiCn6KSBqR9WjXMO7sDtfw2Kd
mdU8h7GCdu7vQ5WRjmBQIylZz+/z1j/CdzUzcdYm/AmWvbub+C+bFs7rLx/tsEFu
luHCFCt/EvYhVmdXAgMBAAGjggJ7MIICdzAMBgNVHRMBAf8EAjAAMIICZQYDVR0O
BIICXASCAliMwRd9fnEZVzLjs+MAAAAAyj3mOOm6Hn8OTxgzYoZf1WY35EkOHSBP
gnweHa0jYc2RxtaTkqodRbM+AzCwpEmMXXRa2737/C41WHAZbzCZEX5Aw70C3LBZ
```

```
rO9sk+4bDWmSEFyg1H2Tl/8386W2Iwj+K/T7YGJ5iU2lDFPGmrcSm8xt66JOzwz8
X21vceW1+8CV13CydSQ0kI+QSB7od6xtZVO3/KFNYIGjTtWTGLxooOzkWsYqO7ft
orIePSe/dXiRczmma/2bFSNFMgBoScveLNKR+5M2MUnCdl+mbdMHFAzh7wq6TT8F
BjkfgY+O5UgS0F6BkMgFru5G4Tj40Zw0ext3iYL8son00oC9tnEqM2QJaIVDVTkz
f5qqbcl3pe7oVFt3O7N4soGyI8twJM7X857NVcpRrGzynN5YegsZl/vxYBfDFZjM
DmIscBTE38A9jfH7ROUkGHBy/u7PSna9nPWgICPjnV7BmCkWxTNA3+gFFmdgGxpl
i6xm2jCnoaqUAkvkqrrYphEdBAhoVjqhYpte+quw+cNuDAdGo+edc3WHhr9/IGOE
2GL1Yt1YZymf3pwdgTm2ybtUCUFlJQMvuwOt0zHXziycj7DGLArZhHn2iaBBSOJn
bALUDJN7L2z6lDM2ZIpms2njV0vLnKQ4+6jGPaJTXmi/DoFNS9kNgSE5etq+mlFt
kVAAEODNM8cStJ2oyXkWoNtfvpbOMPs+mvEz6wSYsDtuCgAx4M72P7fXQYfnBGA+
yiBCaxSYC30W+gorzX+3m7hkzdzusjQKXM5ODR8eT9owDQYJKoZIhvcNAQELBQAD
ggEBABk1VY+CmYV1kcigRGW78zaWXcoRqfDxvzHO2XpNoMcbrLhwcKaIFrsjvKga
SQTiOrVz64z4ZMx6xBtolgT4LljoiCXdnx+7p9TYkV+zLRiaTYRN+NPAiRLjspMU
+O8pUpP6kB33P8u7qSwfrFx4CF5mcVKgPtTsDv2jCkabYUMw+R10Tp5PBXxuvsJn
cVYkrMYG9gx+e9tcmZeRjpN7h+HlhjN8xS4v8sejplSqeslRKFcluZJvHilwYsDx
K+yR8NjLxj5jNbV9PkXvbDl8UetSPu8Z7LVBmvYVsqpT9EMDHny8hwKaj2hhIaC5
XSi2f8lN6a8ZgDB7fNdjrQen6ZQ=
-----END CERTIFICATE-----
```

issuer1.pem

```
-----BEGIN CERTIFICATE-----
MIICoDCCAYgCCQD60kjmTUQwoTANBgkqhkiG9w0BAQsFADASMRAwDgYDVQQDDAdE
ZW1vQ0ExMB4XDTE4MDEyMzAwMzgzMVoXDTI4MDEyMTAwMzgzMVowEjEQMA4GA1UE
AwwHRGVtb0NBMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMX7H7ye
KxVh25Pi7VUQKtgOk8f2GE+/GC0Huvp472/+WHwfqJtAbHJ4kwDVSvZqPMBVTqhK
nhKKxJH6TcFV6gCTU9sioTn8WE6Z4WzY8UI8kIod/Ab3cWZ6+FGAf8TxgrHAPmb3
VgTsOZhPdTiBQyc1MgS+3CMRFO3qoEo2rD+doO8y+q80ljK0algOQVXk8anYDUGW
fbGxw9EKRhsWBogdUn58QVqiCz/y3uuHInnN/4XAGh6heIp0rlisyQs80JDhnupI
afl28KH00t01KiD5QGfMSjcfdQW4JHkOpXCKMGSGw2hvEHgktu/rH5HonXxeqRZs
maa/mGIV9UDC/XsCAwEAATANBgkqhkiG9w0BAQsFAAOCAQEAQmK7/cMCOPEBnbIw
W8cSqAtiUQq8xbElylhMQFCqy8GPQaFYcd2uY1j+emVQhPRnqokZGdIEvZnsoPfZ
lZYYsFSiwDAG6/MKJncz0x6UcSzA0hWQZ2iSlYQ7WfwaujMJ2p6jKXRIRHdPhVBY
THp9hJv7d03hef7mXe4whxHIhS+34YoteMUribVBqqQoMYlQ339XrAqNnHcTP+3c
+ZGFk6mMrQuRuu7HXp/yYYiPf8Hxdzd9HpUy4eg6ztuWww4FL8UpJtLmppwahX01
b1l6HR7Y/Oj2xl7CI9xyS4oFMH1AybhavnooLXIAokumdJ/JJHAInSiSw90euPNL
3s283Q==
-----END CERTIFICATE-----
```

cert2.pem

```
-----BEGIN CERTIFICATE-----
MIIFUDCCBDigAwIBAgIRAP9mhIkfyuhe6JyLNYqGrvMwDQYJKoZIhvcNAQELBQAw
EjEQMA4GA1UEAwwHRGVtb0NBMjAiGA8yMDE4MDIxNDEyMDAwMFoYDzIwMTkwMjE0
MTIwMDAwWjAyMRowGAYDVQQLDBFEZW1vIENlcnRpZmljYXRlMjEUMBIGA1UEAwwL
ZWcyMjIyMi5vcmcwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDVxV70
nmM10QAbHiUVUKc2fTPxl4SY7WwvcUCYesdlzCsEfUGkPP41NGou9H6bdqEk9plj
34RNaT0hHFD4ikWdKV5L4AQZRNjy5iWENBPBpBHaYBqicH8kd33lbB9WCLuah79C
MkE0zykiS4muQSRcxzWuNB/lH9h8eUoCTiJtpfCzEuipLyxtHO/pI120rKJoMIem
qPliHuLxk3i1xv3O8kb8zfV03eFTVNjuxWZYFunjIH3oR+DsXLkoBLr+XY2JIs22
t/bcRsdg8hZQEu4RWh0mKOpiroG5NcqTEUCTlvephwpddxCbyeEUzkj77WuOxMZj
eUKPOI9aYvgBXxblAgMBAAGjggJ7MIICdzAMBgNVHRMBAf8EAjAAMIICZQYDVR0O
```

```
BIICXASCAlh9iZnicV36N5XO6xoAAAAAtOBSGWrueA4OTxgzYoZf1WY35EkOHSBP
gnweHaOjYc2RxtaTkqodRbM+AzCwpEmMXXRa2737/CO1WHAZbzCZEX5Aw7OC3LBZ
rO9sk+4bDWmSEFyg1H2Tl/8386W2Iwj+K/T7YGJ5iU2lDFPGmrcSm8xt66JOOwz8
X21vceW1+8CV13CydSQOkI+QSB7od6xtZVO3/KFNYIGjTtWTGLxooOzkWsYqO7ft
orIePSe/dXiRczmma/2rFSNFMgBoScveLNKR+5M2MUnCdl+mbdMHFAzh7wq6TT8F
BjkfgY+O5UgSOF6BkMgFru5G4Tj4OZwOext3iYL8ron0OoC9tnEqM2QJaIVDVTkz
f5qqbcl3pe7oVFt3O7N4soGyI8twJM7X857NVcpRrGzynN5YegsZl/vxYBfBFZjM
DmIscBTE38A9jfH7ROUkGHBy/u7PSna9nPWgICPjnV7BmCkWxTNA3+gFFmdgGxpl
i6xm2jCnoaqUAkvkorrYphEdBAhoVjqhYpte+quw+cNuDAdGo+edc3WHhr9/IGOE
2GL1Yt1YZymf3pwdgTm2ybtUCUFlJQMvuwOt0zHXzqycj7DGLArZhHn2iaBBSOJn
bALUDJN7L2z6lDM2ZIpms2njV0vLnKQ4+6jGPaJTXmi/DoFNS9kNgSE5eto+mlFt
kVAAEODNM8cStJ2oyXkWoNtfvpbOMPs+mvEz6wSYsDtuCgAx4M72P7fXQYfnBGA+
yiBCaxSYC3OW+gorzX+3i7hkzdzusjQKXM5ODR8eT9owDQYJKoZIhvcNAQELBQAD
ggEBABk1VY+CmYV1kcigRGW78zaWXcoRqfDxvzHO2XpNoMcbrLhwcKaIFrsjvKga
SQTiOrVz64z4ZMx6xBtolgT4LljoiCXdnx+7p9TYkV+zLRiaTYRN+NPAiRLjspMU
+O8pUpP6kB33P8u7qSwfrFx4CF5mcVKgPtTsDv2jCkabYUMw+R10Tp5PBXxuvsJn
cVYkrMYG9gx+e9tcmZeRjpN7h+HlhjN8xS4v8sejplSqeslRKFcluZJvHilwYsDx
K+yR8NjLxj5jNbV9PkXvbDl8UetSPu8Z7LVBmvYVsqpT9EMDHny8hwKaj2hhIaC5
XSi2f8lN6a8ZgDB7fNdjrQen6ZQ=
-----END CERTIFICATE-----
```

issuer2.pem

```
-----BEGIN CERTIFICATE-----
MIIDnzCCAocCCQDMR/Y/Bs+9TTANBgkqhkiG9w0BAQsFADASMRAwDgYDVQQDDAdE
ZW1vQ0EyMB4XDTE4MDEyNjAxMzQxNFoXDTE5MDEyNjAxMzQxNFowEjEQMA4GA1UE
AwwHRGVtbONBMjCCAiEwDQYJKoZIhvcNAQEBBQADggIOADCCAgkCggEBAINLVsgk
zBCovh0D1wMCHtrGYycbLSg/zO4fW0S8xUO0/BUOJSiC7lmIyfkmlVFSXylemzVI
3JjHpznXJkcBjfZhIRWgK58jd/q5aVo9l6IOWZmJHzuVNiYURdpACRmOyFefTZzj
Ib6H8tz1ABKIjNAe1RgzF3yybfdNxHCHx3MVVhu9hbAXmRhzX35TqHG4g/af3EcK
5t4MH+2heVGOjEYYZOaiAioBMXJXM1E16Blp2QrFhDvDhVh8PTgC6AzAJA3ekQQM
Ex6e2OXxeQwroKchMhXRyRm/qSm9gEc+FTsqFdRFfOFo7QNL1x1RIhdK/wUS+ur3
J2GwYU2mEueqtPECggEAYJOrU8IbAoawCF6uHU+tK3H0Jr4S11Mwd0bBFSlqeVuG
ZqPGuCpIxqgsEFBkmscpKa2HQaiwcxBqxzJPmxFPxbSJ/huFK2av86fvehE/M5AS
6Z+AfEqO6GrIhtQ7Vpj4/csnUbAcyE29W+vHkDWcDDKQKzoUU3MZbhdssIUihQIY
dc4RqdZxOPVWHdLZpJwvuzwQwiksBXVHqGsRZtH7DjLtOjWG85nsMFkSQ30exFVx
CNXQSD+gpaH/sVcnRnJK849e9AGA6CFRzhvsby/vyZ34oWQn6idGSLGs4f9fPXKZ
44W9v347PeUBYEuNcQBMfTApTjEQw1Oj7DxNOIsT/TANBgkqhkiG9w0BAQsFAAOC
AQEAAFq92lUGbdPPD14ClrX2pTHWMiQH74dCZtz9ifshjT/nYZyf2qgzvEOUjE3Q
NAJA4Wc6gyRmrKwdwfXFU1CaFpOKIcWfxifOQxrWyUauBcX0NQ6zmhzXMSNjS2Ok
szbSqjUUWEQ9wNH9VuUGV6yk05mM6TrI2D9+7O8nnzB6dGtQYpJv0tCdFBT5xmwq
T4hwcfOaMQQ/FYOx4lAb94IMf2iMUruBh8NtKkISZdjLXb32wW/Ieky62tNQXqYU
7Ubhurskdsmks5GPvAcODCVTj9ZGJVv19pgd0d/xRfyzNHCBbjr/jR1qyrSoUWFj
Pp5XdcMKoKUsWvfKzvVVWUml0A==
-----END CERTIFICATE-----
```

issuer2.key

```
-----BEGIN RSA PRIVATE KEY-----
MIIFoQIBAAKCAQEAg0tWyCTMEKi+HQPXAwIe2sZjJxstKD/M7h9bRLzFQ7T8FQ4l
KILuWYjJ+SaVUVJfKV6bNUjcmMenOdcmRwGN9mEhFaArnyN3+rlpWj2Xog5ZmYkf
O5U2JhRF2kAJGY7IV59NnOMhvofy3PUAEoiM0B7VGDMXfLJt9O3EcIfHcxVWG72F
```

```
sBeZGHNfflOocbiD9p/cRwrm3gwf7aF5UY6MRhhnRqICKgExclczUTXoGWnZCsWE
O8OFWHw9OALoDMAkDd6RBAwTHp7Y5fF5DCugpyEyFdHJGb+pKb2ARz4VOyoV1EV8
4WjtA0vXHVEiF0r/BRL66vcnYbBhTaYS56q08QKCAQBgk6tTwhsChrAIXq4dT60r
cfQmvhLXUzB3RsEVKWp5W4Zmo8a4KkjGqCwQUGSaxykprYdBqLBzEGrHMk+bEU/F
tIn+G4UrZq/zp+96ET8zkBLpn4B8So7oasiG1DtWmPj9yydRsBzITb1b68eQNZwM
MpArOhRTcxluF2ywhSKFAhh1zhGp1nE49VYd0tmknC+7PBDCKSwFdUeoaxFm0fsO
MuO6NYbzmewwWRJDfR7EVXEI1dBIP6Clof+xVydGckrzj170AYDoIVHOG+xvL+/J
nfihZCfqJ0ZIsazh/189cpnjhb2/fjs95QFgS41xAEx9MClOMRDDU6PsPE3QixP9
AoIBAC24+EXEk38nghynzCtTvn4JrXUU8r4MGAlC+aB35xVprLi13Cnz7jDdSV7X
ARHakpujWryavS2dTRJaS0yvqWmhGv2bNbZjtOGsBfRZeMpJDA51U2UO64kRH23d
GxfigaCo7tTrQEjWLcVqJysX5I+gGI5Uj+WL/U1qcU6mTYtrgc9dslR+mHP4Lwco
DOfvz0qzeMyhlPIN4U+91y7rILkF6o6DIMMOmu2qbsCG9yZ2btHinPEBw1U2e8TR
fEGRePRLReFr6S6Inei9gmObATCpJGtpt8r6FMXaHbqLzaWOnULW1IWveDh+9zH7
3htzs4/AyELf5Pna+4EDjDp+e2kCgYEAr+uc4+rAaDaR3JFbWoWPnK2xrliRLa0w
f7yyHa0JnJ9IoOu373w8J/qOnzOwG0vglrOKj5ZnhCMlrMHPWbZwszrb1Cxh21EF
lZvqpEN55W5vRRqs+r1LB1t61hpDs/jT6kJ2VS1GaooaZJO8yY728ilzsIaujuQX
47n8uj9nQJMCgYEAvw9aG8CCDjAPbHOLLHZXgmNJzMWWqJDWpwYuZsyy6dSUx2SA
tnM1iQR6oL82lhzVMKEM1jfeICuYt8iMspWoV/vgUyerUY1d/pU6AfHhBJy0HqLQ
a7euSyOMxhDmwhcPo/tWMIK2dFur5zXNJ5QeNUoRAcsBzNY1MY4TBxUOmusCgYAc
DfHzY7owZEKdcMwvY69d5UAVUM580ks5TWyFo5oilVwnMbSqDZ+Win+2/loS4QfF
HpbfnjJSkhSQGaMDVbak68tnPqKoyexTCFpl0lcCv6IYADGZDzEWii1qWg9HJEGs
0RoQNYdv6MztVpiracViZK2NGCJX2qkTcoXur8d60QKBgEfaUGfZMyDvT9Bmg/q6
fdFx0JTw/dEPeNjRH8ySxfnn8Vb13EE5ipLUyUylhzng4XCY7M0hypWm5MgaevCe
vC22vPeDjyluug4nPtY55s9scrZbMX3+XbrqZpGkFBGZoRX6/WKJpqKAyo8/6wAd
Ex3yzMD571PTS9n6T+PzfZDfAoGAZQ+TV+Xy5PUS9vKRmLep5GuLfEIKg0GJ44oW
6RUfAfER9+J+d6+Ry6OVq1XNzIKcqUlrHHxo3uWc9UklLOjC5FbBtfKZ3JkilVZb
ntfEsUlk+TYCE9uS/I/J3xyjxF93pJYzSDvvCPiCrWPxO/YEHGDcoZH0pE8Qghch
dCnGtNU=
-----END RSA PRIVATE KEY-----
```

B Example Certificates with Mauled Signatures

In this appendix we provide examples of pairs of certificates with identical TBSCertificate data, but different thumbprints, because of differences in the signatures.

B.1 A Mauled ECDSA Certificate

The first example is an ECDSA-signed certificate (`ecdsa.pem`), where we replaced s with $-s \pmod{n}$ (`ecdsa-modified.pem`). The issuer chain is provided as `ecdsa-issuer.pem`.

To view the certificate data and the thumbprints use the commands

```
openssl x509 -fingerprint -noout -in ecdsa.pem -text
openssl x509 -fingerprint -noout -in ecdsa-modified.pem -text
```

The SHA-1 fingerprints should (respectively) be

```
5F1B5BDDA91826A48C86F942D673C3A2C5DF0F0E
66E0E8D044A5D30187E7203279CC98FD95F0ED8F
```

To verify the signatures, use the commands

```
openssl verify -CAfile ecdsa-issuer.pem ecdsa.pem
openssl verify -CAfile ecdsa-issuer.pem ecdsa-modified.pem
```

ecdsa.pem

```
-----BEGIN CERTIFICATE-----
MIIDvDCCA2KgAwIBAgIQDDJLjLyKRnWxamz96kxwsTAKBggqhkjOPQQDAjBvMQsw
CQYDVQQGEwJVUzELMAkGA1UECBMCQ0ExFjAUBgNVBAcTDVNhbiBGcmFuY2lzY28x
GTAXBgNVBAoTEENsb3VkRmxhcmUsIEluYy4xIDAeBgNVBAMTF0Nsb3VkRmxhcmUg
SW5jIEVDQyBDQS0yMB4XDTE4MDQxNjAwMDAwMFoXDTE5MDQxNjEyMDAwMFowazEL
MAkGA1UEBhMCVVMxCzAJBgNVBAgTAkNBMRYwFAYDVQQHEw1TYW4gRnJhbmNpc2Nv
MRkwFwYDVQQKExBDbG91ZEZsYXJlLCBJbmMuMRwwGgYDVQQDExNibG9nLmNsb3Vk
ZmxhcmUuY29tMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEwbh/4/iTgiI21QDC
V8tiMV+ER57/KlCd9Z86xR+OQ5OjSgrAjkgKX1l3YavZ0dNuUZoHSl+YlLOjcxCw
RXIYbqOCAeIwggHeMB8GA1UdIwQYMBaAFD50LR/PRXUEfj/Aooc+TEODURPGMB0G
A1UdDgQWBBSoHQHK06p+BfZiDQv8Wm64fFJghTAeBgNVHREEFzAVghNibG9nLmNs
b3VkZmxhcmUuY29tMA4GA1UdDwEB/wQEAwIHgDAdBgNVHSUEFjAUBggrBgEFBQcD
AQYIKwYBBQUHAwIweQYDVR0fBHIwcDA2oDSgMoYwaHR0cDovL2NybDMuZGlnaWNl
cnQuY29tL0Nsb3VkRmxhcmVJbmNFQ0NDQTIuY3JsMDagNKAyhjBodHRwOi8vY3Js
NC5kaWdpY2VydC5jb20vQ2xvdWRGbGFyZUluY0VDQ0NBMi5jcmwwTAYDVR0gBEUw
QzA3BglghkgBhv1sAQEwKjAoBggrBgEFBQcCARYcaHR0cHM6Ly93d3cuZGlnaWNl
cnQuY29tL0NQUzAIBgZngQwBAgIwdgYIKwYBBQUHAQEEajBoMCQGCCsGAQUFBzAB
hhhodHRwOi8vb2NzcC5kaWdpY2VydC5jb20wQAYIKwYBBQUHMAKGNGh0dHA6Ly9j
YWNlcnRzLmRpZ2ljZXJ0LmNvbS9DbG91ZEZsYXJlSW5jRUNDQ0EtMi5jcnQwDAYD
VR0TAQH/BAIwADAKBggqhkjOPQQDAgNIADBFAiEAkrdDjkReXxAvANnf0vIiNx86
Ef9G86Ybmr++QyjB8JcCIA/ON3/DYPqblrQRYqXd3QjqXbPSIGs34PuWWC2jcGFw
-----END CERTIFICATE-----
```

ecdsa-modified.pem

```
-----BEGIN CERTIFICATE-----
MIIDvTCCA2KgAwIBAgIQDDJLjLyKRnWxamz96kxwsTAKBggqhkjOPQQDAjBvMQsw
CQYDVQQGEwJVUzELMAkGA1UECBMCQ0ExFjAUBgNVBAcTDVNhbiBGcmFuY2lzY28x
GTAXBgNVBAoTEENsb3VkRmxhcmUsIEluYy4xIDAeBgNVBAMTF0Nsb3VkRmxhcmUg
SW5jIEVDQyBDQS0yMB4XDTE4MDQxNjAwMDAwMFoXDTE5MDQxNjEyMDAwMFowazEL
MAkGA1UEBhMCVVMxCzAJBgNVBAgTAkNBMRYwFAYDVQQHEw1TYW4gRnJhbmNpc2Nv
MRkwFwYDVQQKExBDbG91ZEZsYXJlLCBJbmMuMRwwGgYDVQQDExNibG9nLmNsb3Vk
ZmxhcmUuY29tMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEwbh/4/iTgiI21QDC
V8tiMV+ER57/KlCd9Z86xR+OQ5OjSgrAjkgKX1l3YavZ0dNuUZoHSl+YlLOjcxCw
RXIYbqOCAeIwggHeMB8GA1UdIwQYMBaAFD50LR/PRXUEfj/Aooc+TEODURPGMB0G
A1UdDgQWBBSoHQHK06p+BfZiDQv8Wm64fFJghTAeBgNVHREEFzAVghNibG9nLmNs
b3VkZmxhcmUuY29tMA4GA1UdDwEB/wQEAwIHgDAdBgNVHSUEFjAUBggrBgEFBQcD
AQYIKwYBBQUHAwIweQYDVR0fBHIwcDA2oDSgMoYwaHR0cDovL2NybDMuZGlnaWNl
cnQuY29tL0Nsb3VkRmxhcmVJbmNFQ0NDQTIuY3JsMDagNKAyhjBodHRwOi8vY3Js
NC5kaWdpY2VydC5jb20vQ2xvdWRGbGFyZUluY0VDQ0NBMi5jcmwwTAYDVR0gBEUw
QzA3BglghkgBhv1sAQEwKjAoBggrBgEFBQcCARYcaHR0cHM6Ly93d3cuZGlnaWNl
cnQuY29tL0NQUzAIBgZngQwBAgIwdgYIKwYBBQUHAQEEajBoMCQGCCsGAQUFBzAB
hhhodHRwOi8vb2NzcC5kaWdpY2VydC5jb20wQAYIKwYBBQUHMAKGNGh0dHA6Ly9j
```

```
YWNlcnRzLmRpZ2ljZXJ0LmNvbS9DbG91ZEZsYXJlSW5jRUNDQ0EtMi5jcnQwDAYD
VR0TAQH/BAIwADAKBggqhkjOPQQDAgNJADBGAiEAkrdDjkReXxAvANnf0vIiNx86
Ef9G86Ybmr++QyjB8JcCIQDwMch/PJ8FZWlL7p1aIiL20olG24asZqP4I3KVWPLD
4Q==
-----END CERTIFICATE-----
```

ecdsa-issuer.pem

```
-----BEGIN CERTIFICATE-----
MIIDdzCCAl+gAwIBAgIEAgAAuTANBgkqhkiG9w0BAQUFADBaMQswCQYDVQQGEwJJ
RTESMBAGA1UEChMJQmFsdGltb3JlMRMwEQYDVQQLEwpDeWJlclRydXN0MSIwIAYD
VQQDExlCYWx0aW1vcmUgQ3liZXJUcnVzdCBSb290MB4XDTAwMDUxMjE4NDYwMFoX
DTI1MDUxMjIzNTkwMFowWjELMAkGA1UEBhMCSUUxEjAQBgNVBAoTCUJhbHRpbW9y
ZTETMBEGA1UECxMKQ3liZXJUcnVzdDEiMCAGA1UEAxMZQmFsdGltb3JlIEN5YmVy
VHJ1c3QgUm9vdDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAKMEuyKr
mD1X6CZymrV51Cni4eiVgLGw41uOKymaZN+hXe2wCQVt2yguzmKiYv60iNoS6zjr
IZ3AQSsBUnuId9Mcj8e6uYi1agnnc+gRQKfRzMpijS3ljwumUNKoUMMo6vWrJYeK
mpYcqWe4PwzV9/lSEy/CG9VwcPCPwBLKBsua4dnKM3p31vjsufFoREJIE9LAwqSu
XmD+tqYF/LTdB1kC1FkYmGP1pWPgkAx9XbIGevOF6uvUA65ehD5f/xXtabz5OTZy
dc93Uk3zyZAsuT3lySNTPx8kmCFcB5kpvcY67Oduhjprl3RjM71oGDHweI12v/ye
jl0qhqdNkNwnGjkCAwEAAaNFMEMwHQYDVR0OBBYEFOWdWTCCR1jMrPoIVDaGezq1
BE3wMBIGA1UdEwEB/wQIMAYBAf8CAQMwDgYDVR0PAQH/BAQDAgEGMA0GCSqGSIb3
DQEBBQUAA4IBAQCFDF2O5G9RaEIFoN27TyclhAO992T9Ldcw46QQF+vaKSm2eT92
9hkTI7gQCvlYpNRhcL0EYWoSihfVCr3FvDB81ukMJY2GQE/szKN+OMY3EU/t3Wgx
jkzSswF07r51XgdIGn9w/xZchMB5hbgF/X++ZRGjD8ACtPhSNzkE1akxehi/oCr0
Epn3o0WC4zxe9Z2etciefC7IpJ5OCBRLbf1wbWsaY71k5h+3zvDyny67G7fyUIhz
ksLi4xaNmjICq44Y3ekQEe5+NauQrz4wlHrQMz2nZQ/1/I6eYs9HRCwBXbsdtTLS
R9I4LtD+gdwyah617jzV/OeBHRnDJELqYzmp
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
MIIDozCCAougAwIBAgIQD/PmFjmqPRoSZfQfizTltjANBgkqhkiG9w0BAQsFADBa
MQswCQYDVQQGEwJJRTESMBAGA1UEChMJQmFsdGltb3JlMRMwEQYDVQQLEwpDeWJl
clRydXN0MSIwIAYDVQQDExlCYWx0aW1vcmUgQ3liZXJUcnVzdCBSb290MB4XDTE1
MTAxNDEyMDAwMFoXDTIwMTAwOTEyMDAwMFowbzELMAkGA1UEBhMCVVMxCzAJBgNV
BAgTAkNBMRYwFAYDVQQHEw1TYW4gRnJhbmNpc2NvMRkwFwYDVQQKExBDbG91ZEZs
YXJlLCBJbmMuMSAwHgYDVQQDExdDbG91ZEZsYXJlIEluYyBFQ0MgQ0EtMjBZMBMG
ByqGSM49AgEGCCqGSM49AwEHA0IABNFW9Jy25DGg9aRSz+Oaeob/8oayXsy1WcwR
x07dZP1VnGDjoEvZeFT/SFC6ouGhWHWPx2A3RBZNVZns7tQzeiOjggEZMIIBFTAS
BgNVHRMBAf8ECDAGAQH/AgEAMA4GA1UdDwEB/wQEAwIBhjA0BggrBgEFBQcBAQQo
MCYwJAYIKwYBBQUHMAGGGGh0dHA6Ly9vY3NwLmRpZ2ljZXJ0LmNvbTA6BgNVHR8E
MzAxMC+gLaArhilodHRwOi8vY3JsMy5kaWdpY2VydC5jb20vT21uaXJvb3QyMDI1
LmNybDA9BgNVHSAENjA0MDIGBFUdIAAwKjAoBggrBgEFBQcCARYcaHR0cHM6Ly93
d3cuZGlnaWNlcnQuY29tL0NQUzAdBgNVHQ4EFgQUPnQtH89FdQR+P8Cihz5MQ4NR
E8YwHwYDVR0jBBgwFoAU5Z1ZMIJHWMys+ghUNoZ7OrUETfAwDQYJKoZIhvcNAQEL
BQADggEBADhfp//8hfJzMuTVo4mZlmCvMsEDs2Xfvh4DyqXthbKPr0uMc48qjKkA
DgEkF/fsUoV2yOUcecrDF4dQtgQzNp4qnhgXljISr0PMVxje28fYiCWD5coGJTH9
vV1IO1EB3SwUx8FgUemVAdiyM1YOR2aNbM2v+YXZ6xxHR4g06PD6wqtPaU4JWdRX
xszByOPmGcFYOFLi4oOF3iIO3D+m968kvOBvwKtoLVLHawVXLEIbLUiHAwyQq0hI
qSi+NIr7uu30YJkdFXgRqtltU39pKLy3ayB2f6BVA3F59WensKAKF1eyAKmtz/9n
jD4m5ackvMJvEOiJxnCl0h+A7Q0/JxM=
```

```
-----END CERTIFICATE-----
```

B.2 A Mauled RSA Certificates

The second example demonstrates mauling the signatureAlgorithm field in the Certificate ASN.1 sequence. Two certificates are provided: `cert-has-NULL.pem` a certificate with the parameters field set explicitly to NULL (with the bytes `05 00`), and `cert-no-NULL.pem` where the parameters field is omitted. The issuer certificate is `issuer-NULL.pem`. Verification of `cert-no-NULL.pem` will fail in OpenSSL. When opening the certificates in Windows (save them as `.crt` files), they will both verify correctly, and have the thumbprints:

```
50f125efca2428ff17d204b89e5264509a283da7
4736619c340b3b28979fa857039f8b2d7ff82c8e
```

cert-has-NULL.pem

```
-----BEGIN CERTIFICATE-----
MIIE3DCCA8SgAwIBAgIQPiMOWu0sClF7Jt7UgBOQqjANBgkqhkiG9w0BAQsFADCB
rjELMAkGA1UEBhMCVVMxFTATBgNVBAoTDHRoYXd0ZSwgSW5jLjEoMCYGA1UECxMf
Q2VydGlmaWNhdGlvbiBTZXJ2aWNlcyBEaXZpc2lvbjE4MDYGA1UECxMvKGMpIDIw
MDggdGhhd3RlLCBJbmMuIC0gRm9yIGF1dGhvcml6ZWQgdXNlIG9ubHkxJDAiBgNV
BAMTG3RoYXd0ZSBQcmltYXJ5IFJvb3QgQ0EgLSBHMzAeFw0xNDA2MTAwMDAwMDBa
Fw0yNDA2MDkyMzU5NTlaMGUxCzAJBgNVBAYTAlVTMRUwEwYDVQQKEwx0aGF3dGUs
IEluYy4xHTAbBgNVBAsTFERvbWFpbiBWYWxpZGF0ZWQgU1NMMSAwHgYDVQQDExd0
aGF3dGUgRFYgU1NMIFNIQTI1NiBDQTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCC
AQoCggEBALOsDX+tuxNNlF9nQmrQiXGp7XQEkyTITVah8JGWhNmEas9SIeMasVRM
5saenks4qZZUHfWz7ZIEOG5UkG4v6X2YtIotEqOOQkcdf19A4fx/kaYB3FWkUHgq
Yz+EfizIKyG2xg5evLix1BuYs8b44ego7TJEG8t/9+SxEevGCLBb7qjC7Eaqjynf
ubekA6A1elg/iylHwdIi+izGx2zNO/dYMpOUOW+pKpwPCiiSqxQKtt/tQHpkB1TO
6nWXMrmWoHXJdzECdK9Ud0+ZooFLeVm4kj/5B+pCdFcuNexVivxhPD5XcZI7q+TB
4RcsZDYAhLV8Gn2wQTN8I/ZOd1oswUsCAwEAAaOCATwwggE4MC4GCCsGAQUFBwEB
BCIwIDAeBggrBgEFBQcwAYYSaHR0cDovL3Quc3ltY2QuY29tMBIGA1UdEwEB/wQI
MAYBAf8CAQAwQQYDVR0gBDowODA2BgpghkgBhvhFAQc2MCgwJgYIKwYBBQUHAgEW
Gmh0dHBzOi8vd3d3LnRoYXd0ZS5jb20vY3BzMDQGA1UdHwQtMCswKaAnoCWGI2h0
dHA6Ly90LnN5bWNiLmNvbS9UaGF3dGVQQ0EtRzMuY3JsMA4GA1UdDwEB/wQEAwIB
BjApBgNVHREEIjAgpB4wHDEaMBgGA1UEAxMRU3ltYW50ZWNQS0ktMS02OTUwHQYD
VR0OBBYEFH0pMS/BHm6uMQVqs+sczandroCaMB8GA1UdIwQYMBaAFK1sqpRgnO3k
//o+CnQrYwP3tlm/MA0GCSqGSIb3DQEBCwUAA4IBAQA2/6LxHH65UXuU01p7SCXT
N6KCKi1fOB6HZ+zJMavXkjO4vTXKsYBwBIJ8iMw3LhZ0bpNAY8qNe/8HKOb5M6vw
YYO9yoPFUNi9aTkfrry37hXFjQQGIDMoBJnFnBH1AQ9HXtiJmaXOwoD+Rvrvthuo
kbKDs+JXDRrkltW8971tA/hifuv4Qgn+CWSkyVy40jkLeQKeFTkdwNnNHF9odo3z
Hi36v6dJog2X9ZbC6WzUzUcLi4oBi9v6z5J1Lt4+p3O1/gNRp0LDx0JrqW++9iDh
jr+fCY7lCOiSk3c+SUScf+l5nf9Lr+A4VzQNXxEyEpKpYYiBpR74oPBFWoZxIIWF
-----END CERTIFICATE-----
```

cert-no-NULL.pem

```
-----BEGIN CERTIFICATE-----
MIIE2jCCA8SgAwIBAgIQPiMOWu0sClF7Jt7UgBOQqjANBgkqhkiG9w0BAQsFADCB
```

```
rjELMAkGA1UEBhMCVVMxFTATBgNVBAoTDHRoYXd0ZSwgSW5jLjEoMCYGA1UECxMf
Q2VydGlmaWNhdGlvbiBTZXJ2aWNlcyBEaXZpc2lvbjE4MDYGA1UECxMvKGMpIDIw
MDggdGhhd3RlLCBJbmMuIC0gRm9yIGF1dGhvcml6ZWQgdXNlIG9ubHkxJDAiBgNV
BAMTG3RoYXd0ZSBQcmltYXJ5IFJvb3QgQ0EgLSBHMzAeFw0xNDA2MTAwMDAwMDBa
Fw0yNDA2MDkyMzU5NTlaMGUxCzAJBgNVBAYTAlVTMRUwEwYDVQQKEwx0aGF3dGUs
IEluYy4xHTAbBgNVBAsTFERvbWFpbiBWYWxpZGF0ZWQgU1NMMSAwHgYDVQQDExd0
aGF3dGUgRFYgU1NMIFNIQTI1NiBDQTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCC
AQoCggEBALOsDX+tuxNNlF9nQmrQiXGp7XQEkyTITVah8JGWhNmEas9SIeMasVRM
5saenks4qZZUHfWz7ZIE0G5UkG4v6X2YtIotEqO0Qkcdf19A4fx/kaYB3FWkUHgq
Yz+EfizIKyG2xg5evLix1BuYs8b44ego7TJEG8t/9+SxEevGCLBb7qjC7Eaqjynf
ubekA6A1elg/iylHwdIi+izGx2zN0/dYMpOU0W+pKpwPCiiSqxQKtt/tQHpkB1TO
6nWXMrmWoHXJdzECdK9Ud0+ZooFLeVm4kj/5B+pCdFcuNexVivxhPD5XcZI7q+TB
4RcsZDYAhLV8Gn2wQTN8I/ZOd1oswUsCAwEAAaOCATwwggE4MC4GCCsGAQUFBwEB
BCIwIDAeBggrBgEFBQcwAYYSaHR0cDovL3Quc3ltY2QuY29tMBIGA1UdEwEB/wQI
MAYBAf8CAQAwQQYDVR0gBDowODA2BgpghkgBhvhFAQc2MCgwJgYIKwYBBQUHAgEW
Gmh0dHBzOi8vd3d3LnRoYXd0ZS5jb20vY3BzMDQGA1UdHwQtMCswKaAnoCWGI2h0
dHA6Ly90LnN5bWNiLmNvbS9UaGF3dGVQQ0EtRzMuY3JsMA4GA1UdDwEB/wQEAwIB
BjApBgNVHREEIjAgpB4wHDEaMBgGA1UEAxMRU3ltYW50ZWNQS0ktMS02OTUwHQYD
VR0OBBYEFH0pMS/BHm6uMQVqs+sczandroCaMB8GA1UdIwQYMBaAFK1sqpRgnO3k
//o+CnQrYwP3tlm/MAsGCSqGSIb3DQEBCwOCAQEANv+i8Rx+uVF7lNNae0gl0zei
giotXzgeh2fsyTGr15IzuLO1yrGAcASCfIjMNy4WdG6TQGPKjXv/Byjm+TOr8GGN
PcqDxVDYvWk5H668t+4VxYOEBiAzKASZxZwR9QEPR17YiZmlzsKA/kb677YbqJGy
g7PiVw0a5JbVvPe9bQP4Yn7r+EIJ/glkpMlcuNI5C3kCnhU5HcDZzRxfaHaN8x4t
+r+nSaINl/WWwuls1M1HC4uKAYvb+s+SdS7ePqdztf4DUadCw8dCa6lvvvYg4Y6/
nwmO5QjokpN3PklEnH/peZ3/S6/gOFc0DV8RMhKSqWGIgaUe+KDwRVqGcSCFhQ==
-----END CERTIFICATE-----
```

issuer-NULL.pem

```
-----BEGIN CERTIFICATE-----
MIIEKjCCAxKgAwIBAgIQYAGXt0an6rS0mtZLL/eQ+zANBgkqhkiG9w0BAQsFADCB
rjELMAkGA1UEBhMCVVMxFTATBgNVBAoTDHRoYXd0ZSwgSW5jLjEoMCYGA1UECxMf
Q2VydGlmaWNhdGlvbiBTZXJ2aWNlcyBEaXZpc2lvbjE4MDYGA1UECxMvKGMpIDIw
MDggdGhhd3RlLCBJbmMuIC0gRm9yIGF1dGhvcml6ZWQgdXNlIG9ubHkxJDAiBgNV
BAMTG3RoYXd0ZSBQcmltYXJ5IFJvb3QgQ0EgLSBHMzAeFw0wODA0MDIwMDAwMDBa
Fw0zNzEyMDEyMzU5NTlaMIGuMQswCQYDVQQGEwJVUzEVMBMGA1UEChMMdGhhd3Rl
LCBJbmMuMSgwJgYDVQQLEx9DZXJ0aWZpY2F0aW9uIFNlcnZpY2VzIERpdmlzaW9u
MTgwNgYDVQQLEy8oYykgMjAwOCB0aGF3dGUsIEluYy4gLSBGb3IgYXV0aG9yaXpl
ZCB1c2Ugb25seTEkMCIGA1UEAxMbdGhhd3RlIFByaW1hcnkgUm9vdCBDQSAtIEcz
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsr8nLPvb2FvdeHsbnndm
gcs+vHyu86YnmjSjaDFxODNi5PNxZnmxqWWjpYvVj2AtP0LMqmsywCPLLEHd5N/8
YZzic7IilRFDGF/Eth9XbAoFWCLINkw6fKXRz4aviKdEAhN0cXMKQlkC+BsUa0Lf
b1+6a4KinVvnSr0eAXLbS3ToO39/fR8EtCab4LRarEc9VbjXsCZSKAExQGbY2SS9
9irY7CFJXJv2eul/VTV+lmuNk5Mny5K76qxAwJ/C+IDPXfRa3M50hqY+bAtTyr2S
zhkGcuYMXDhpxwTWvGzOW/b3aJzcJRVIiKHpqfiYnODz1TEoYRFsZ5aNOZnLwkUk
OQIDAQABo0IwQDAPBgNVHRMBAf8EBTADAQH/MA4GA1UdDwEB/wQEAwIBBjAdBgNV
HQ4EFgQUrWyqlGCc7eT/+j4KdCtjA/e2Wb8wDQYJKoZIhvcNAQELBQADggEBABpA
2JVlrAmSicY59BDlqQ5mU1143vokkbvnRFHfxhY0Cu9qRFHqKweKA3rD6z8KLFIW
oCtDuSWQP3CpMyVtRRooOyfPqsMpQhvfO0zAMzRbQYi/aytlryjvsvXDqmbOe1bu
t8jLZ8HJnBoYuMTDSQPxYA5QzUbF83d597YV4Djbxy8ooAw/dyZ02SUS2jHaGh7c
KUGRIjxpp7sC8rZcJwOJ9Abqm+RyguOhCcHpABnTPtRwa7pxpqpYrvS76Wy274fM
```

```
m7v/OeZWYdMKp8RcTGB7BXcmer/YB1IsYvdwY9k5vG8cwnncdimvzsUsZAReiDZu
MdRAGmIONj81Aa6sY6A=
-----END CERTIFICATE-----
```

C Certificate Pinning

Certificate pinning is one of the scenarios we investigated when looking at thumbprint security, so we include a description of pinning and how certificate thumbprints could affect security in some implementations. Admittedly, the scenario is unlikely, and any weakness would be difficult to exploit, but it serves as a concrete example.

Operating systems generally ship with a certificate store, a collection of trusted root CA certificates. Applications like browsers, mail clients and TLS libraries may use these roots when validating certificates. However, since there are a large number of CA certificates in the platform certificate store, applications sometimes wish to limit which certificates are trusted. For example, consider a mobile application that makes connections to a cloud service (operated by the application author). These connections are secured with TLS, and the application must validate the server certificate chain. The Android platform certificate store contains over one hundred root certificates belonging to about seventy CAs.[8] A breach or malicious behavior by any one of these CAs can allow a network attacker to successfully authenticate itself as the application's cloud backend, and obtain user data, and modify data sent to the application. The application can reduce its attack surface by limiting the certificates it trusts. How this is done varies by application and platform.

The application can require that the certificate first chains to a root in the store, then apply further checks. Applications that don't need their backend services to be trusted by browsers can use self-issued certificates and can skip validation against the platform store. The further checks may be one, or a combination of the following:

- Checking that the root certificate is a known value.
- Checking that the certificate chain contains a specific intermediate CA certificate.
- Checking that the end-entity certificate is a specific value.
- Checking that the public key of one certificate in the chain matches a known value.

The known values are referred to as *pins* and they ship as part of the application. In most examples we could find, the pin is a hash of the certificate or key, and the application hashes received certificates and public keys to compare them to the pins. There are many secure ways of implementing pinning [24], with various trade-offs. If the application implements pinning by comparing certificate hashes to known hashes, there is potentially weakness if thumbprints are not unique. If the attacker can obtain a certificate which hashes to the

[8] Checked on Android 8.0 (Oreo), January 2018, on a Google Nexus 5x.

pin, for which he knows the private key, he can pass the pinning check, and authenticate to the application.

Note that in this case, even the CA that issued the server certificate cannot impersonate the server since the application will only accept certificates matching the pin — other certificates the CA creates will be rejected. Therefore, crafting two certificates with the same thumbprint can allow the CA to intercept the app traffic, in a way not possible when thumbprints are unique.

Research related to certificate pinning Chothia et al. [6] and Stone et al. [36] review mobile apps and describe how some (mis)use pining to restrict the root CAs that can authenticate a TLS server. Clark and van Oorschot [7] survey different mechanisms for pinning public keys in TLS connections supported by browsers, and Kranch and Bonneau [16] study how some of these mechanisms are implemented and deployed. Fahl et al. [10] study use of TLS by mobile apps and discuss the challenges with using certificate pinning, and provide tools to help developers. Oltrogge et al. [23] review a large set of Android applications and estimate the feasibility of using certificate pinning, describe implementations, their flaws, and how the Android platform supports pinning.

C.1 Detailed Attack Scenario

Here we explain a specific scenario in greater detail. The attack scenario is a targeted attack, similar to Flame [11], where the attacker is trying to avoid detection.

- The CA is malicious or (more likely) compromised by the attacker. Without this condition, input to the thumbprint algorithm will always end in a large unpredictable value (the signature), and the serial number will be a large unpredictable value, so this seems difficult to avoid.
- All issued certificates will be public, and even logged in systems like Certificate Transparency [17]. The attack should remain undetectable.
- The legitimate subject (owner of certificate C_1) is a high value website, web API, or VPN service. The first certificate C_1 is created between the CA and the subject. From the requester's perspective the process should not differ from the process of interacting with an honest CA. The attacker may not necessarily know the key used to sign C_1, but needs at least oracle access to it when creating C_1.
- We assume that clients (the relying parties) pin the certificate by thumbprint only.
  - E.g., the application ships with a (list of) certificate thumbprint(s), and accepts any TLS connections that complete successfully with an end-entity certificate having the same thumbprint.

  - This is consistent with OWASP guidance on certificate pinning [24].
  - Optionally, the client may require that the pinned certificate chains up to a root in the certificate store. This is discussed in Section 3.

Attacker Goal:

- Given a CSR from the legitimate subject, create a legitimate certificate C_1, and a second certificate C_2 with a chosen public key, such that C_1 and C_2 have the same thumbprint.
- The malicious certificate should be as close to possible as the legitimate certificate, but the public key must be different, ideally the key pair is secure and known only to the attacker, and indistinguishable from honestly generated keys.
- Certificate C_2 can then be used in a man-in-the-middle attack against the client, where the attacker authenticates as the legitimate subject.

Limitations Admittedly this scenario is somewhat contrived and requires a sophisticated attacker. When compared to the Flame attack, a known sophisticated attack on PKI, it additionally requires compromise of one or more CAs, whereas Flame only required interacting with the CA as a subject of the PKI. Since most CAs have migrated away from MD5 and SHA1, it's natural that new cryptographic attacks increase in sophistication.