On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 v1.5 Encryption

Tibor Jager
Horst Görtz Institute
Ruhr University Bochum
tibor.jager@rub.de

Jörg Schwenk
Horst Görtz Institute
Ruhr University Bochum
joerg.schwenk@rub.de

Juraj Somorovsky
Horst Görtz Institute
Ruhr University Bochum
juraj.somorovsky@rub.de

ABSTRACT

Encrypted key transport with RSA-PKCS#1 v1.5 is the most commonly deployed key exchange method in all current versions of the Transport Layer Security (TLS) protocol, including the most recent version 1.2. However, it has several well-known issues, most importantly that it does not provide *forward secrecy*, and that it is prone to *side channel attacks* that may enable an attacker to learn the session key used for a TLS session. A long history of attacks shows that RSA-PKCS#1 v1.5 is extremely difficult to implement securely. The current draft of TLS version 1.3 dispenses with this encrypted key transport method. But is this sufficient to protect against weaknesses in RSA-PKCS#1 v1.5?

We describe attacks which transfer the potential weakness of prior TLS versions to two recently proposed protocols that *do not even support PKCS#1 v1.5 encryption*, namely Google's QUIC protocol and TLS 1.3. These attacks enable an attacker to impersonate a server by using a vulnerable TLS-RSA server implementation as a "signing oracle" to compute valid signatures for messages chosen by the attacker.

The first attack (on TLS 1.3) requires a very fast "Bleichenbacher-oracle" to create the TLS `CertificateVerify` message before the client drops the connection. Even though this limits the practical impact of this attack, it demonstrates that simply removing a legacy algorithm from a standard is not necessarily sufficient to protect against its weaknesses.

The second attack on Google's QUIC protocol is much more practical. It can also be applied in settings where forging a signature with the help of a "Bleichenbacher-oracle" may take an *extremely* long time. This is because signed values in QUIC are *independent* of the client's connection request. Therefore the attacker is able to pre-compute the signature long before the client starts a connection. This makes the attack practical. Moreover, the impact on QUIC is much more dramatic, because creating a single forged signature is essentially equivalent to retrieving the long-term secret key of the server.

CCS'15, October 12–16, 2015, Denver, Colorado, USA.

ACM 978-1-4503-3832-5/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813657.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols; K.4.4 [**Computers and Society**]: Electronic Commerce – Security

Keywords

Cross-protocol attack; chosen-ciphertext attack; TLS

1. INTRODUCTION

TLS and PKCS#1 v1.5 encryption. *Transport layer security* (TLS) is the most important security protocol on the Internet. It is very flexible, as it allows two communicating parties to negotiate the cryptographic algorithms used for a TLS connection at the beginning of each session. For each new TLS session, the peers may choose from different selections of cryptographic algorithms for key establishment, symmetric encryption, and message authentication. The current TLS version is 1.2 [12], it was published in 2008.

All current TLS versions include RSA-based PKCS#1 v1.5 encryption [24]. Even though this scheme has been updated [21, 19] and has been subject to many attacks [7, 22, 9, 18, 4, 28, 36] (in TLS and other applications), it is the most commonly used method for TLS key establishment in practice. Moreover, the only mandatory-to-implement cipher suite in TLS 1.2 is based on this encryption scheme.

TLS 1.3. Version 1.3 of TLS is currently under development [13]. It is the first version which dispenses with PKCS#1 v1.5 encryption. There are several reasons for this, including the lack of *forward secrecy*, that the `PremasterSecret` of previous TLS versions may depend only on client randomness, and that PKCS#1 v1.5 encryption has proven to be *extremely* difficult to implement securely in TLS [7, 22, 4, 28].

In this paper, we analyze the security of the current version of the TLS 1.3 draft against weaknesses in PKCS#1 v1.5 encryption. We show that even though this encryption scheme is not even used in TLS 1.3, the coexistence with older TLS versions may enable *cross-protocol* attacks [33, 26].

Google's QUIC protocol. *Quick UDP Internet Connections* (QUIC) [32] is a key-exchange protocol based on UDP connections, which aims at reducing the relatively high latency of protocols like TLS. The latency in TLS(-over-TCP) stems from the handshake messages (for both TCP and TLS) to be sent before the first encrypted message can be transmitted. QUIC's goal is to reduce the number of "round trips" for key establishment to a minimum, while providing all security guarantees expected from a key-exchange protocol on the Internet.

QUIC is currently considered experimental, but put forward by Google and implemented in recent versions of Google's Chrome web browser, the Opera browser, and available on Google web servers. Google has announced that QUIC will be proposed as an IETF standard.[1] Currently, draft version 01 is available.[2]

Relevance and recurrence of Bleichenbacher attacks. At Crypto 1998 [7] Bleichenbacher presented a seminal attack on RSA-based PKCS#1 v1.5 encryption. Essentially, it assumes the existence of an "oracle" that allows an attacker to distinguish "valid" from "invalid" PKCS#1 v1.5-padded ciphertexts. Such an oracle may in practice be given by a TLS server, which responds with appropriate error messages, or allows in any other way to tell whether a given ciphertext has a "valid" padding or not (for instance by observing the timing behavior of the server when processing the ciphertext). Bleichenbacher showed that such an oracle is sufficient to decrypt PKCS#1 v1.5 ciphertexts. Today there exist many variants and refinements of this attack [7, 22, 9, 18, 4, 28, 36], the TLS RFCs [10, 11, 12] give recommendations for implementing PKCS#1 v1.5 encryption securely (which are however kept quite general).

One may think that seventeen years after the publication of Bleichenbacher's attack we should have developed a very good understanding of this weakness, and that at least important, widely-used applications that still use this scheme (possibly for legacy reasons, like TLS) finally implement it securely. However, there is vast evidence that this belief is false:

1. Increasingly sophisticated analysis techniques and new side channels allow one to apply Bleichenbacher-like attacks in settings where these attacks were previously believed to be impossible. A recent example was described at USENIX Security 2014 by Meyer et al. [28]. The authors discover new timing-based side channels that allow one to apply Bleichenbacher's attack to widely-used TLS implementations, like Java's JSSE or hardware security appliances using Cavium's Nitrox SSL accelerator. Similar new attack techniques are developed perpetually. Further examples include attacks on Europay-Mastercard-Visa (EMV) [9], XML Encryption [18], and more attacks on TLS implementations [22, 4].
2. New applications provide new opportunities to adversaries, by enabling side channels and attack techniques that have not been considered before. For instance, at ACM CCS 2014, Zhang et al. [36] showed very efficient Bleichenbacher-type attacks in Platform-as-a-Service (PaaS) applications. These attacks exploit the fact that, due to virtualization of machines, an attacker may have access to the same physical resources as the victim, which is different from the classical network attacker model. Even though the application considered in [36] implemented all existing countermeasures against Bleichenbacher's attack, and thus was considered not exploitable, the new attack technique circumvents all these countermeasures.

   Please note also that the efficiency of a Bleichenbacher attack may depend on the attacker model: In [28], a network attacker model was used against `OpenSSL`. An answer from the oracle could only be used with low probability 2^{-40} to advance the Bleichenbacher attack by one step. In contrast, Zhang et al. [35] achieve a significantly better success probability of roughly 2^{-16} against *the same* `OpenSSL` version, because their *cross-tenant* attacker model allowed the attacker to "look inside the PKCS#1 v1.5-checking machine".
3. While Bleichenbacher's attack and its early applications [7, 22] were originally rather inefficient, much faster versions are known today [25, 4]. This makes such attacks applicable in settings where they were previously considered infeasible.

Thus, in summary, PKCS#1 v1.5 encryption has proven to be *extremely* difficult to implement securely. Considering *all* possible side channels (some of which might not even be conceivable today) seems virtually impossible.

Transferring the weaknesses of PKCS#1 v1.5 encryption to modern protocols. For the reasons explained above, it seems a wise decision that PKCS#1 v1.5 encryption is not used in QUIC and will finally be removed from TLS in version 1.3. However, the question that motivates our research is the following:

Is this sufficient to protect TLS 1.3 and QUIC against the weaknesses of PKCS#1 v1.5 encryption?

We analyze the security of both protocols under the hypothesis that Bleichenbacher-like attacks on PKCS#1 v1.5 encryption in TLS versions prior to 1.3 will remain a realistic threat in the future, and study the impact of such attacks on TLS 1.3 and QUIC.

For the analysis of TLS 1.3, we consider a setting where there is a TLS client C that supports *only* TLS 1.3, and thus may expect that it is immune against weaknesses in PKCS#1 v1.5 encryption. The client connects to a server S, which offers TLS 1.3, and at least one previous TLS version which allows to use PKCS#1 v1.5 encryption, say TLS 1.2. Note that today's TLS servers are typically not configured to offer only the most recent TLS version 1.2 (or any other *single* version), but they usually offer many TLS versions in parallel, to maximize compatibility with different TLS clients. According to SSL Pulse [1], TLS 1.2 and 1.1 are supported by about 60%, and TLS 1.0 is supported by nearly 100% of the TLS servers analyzed in April 2015. We consider a setting where the server uses the same RSA certificate for both TLS versions.[3] We show that a vulnerability of the old TLS version against Bleichenbacher's attack gives rise to a man-in-the-middle attack on TLS 1.3, which allows an attacker to impersonate S towards C.

The attack (see Figure 1) is based on the observation that Bleichenbacher's attack enables the attacker to perform an RSA secret-key operation *without* knowing the secret RSA key. It exploits that this is sufficient to compute a "forged" RSA signature, which in turn is sufficient to impersonate S towards C in TLS 1.3 (and also in previous TLS versions). A similar technique was used in [9, 17] to compute forged RSA signatures, but to attack different applications. See Section 6 for a detailed description.

The analysis of QUIC is nearly identical. That is, we consider a setting with a client which implements *only* QUIC (and thus may assume to be immune against Bleichenbacher-attacks on TLS), a server which implements QUIC and some TLS version ≤ 1.2, say for interoperability or backwards-compatibility reasons, and where TLS server uses an RSA certificate.

The potentially devastating impact on QUIC. A Bleichenbacher-attack against a TLS server prior to version 1.3 usually allows to decrypt the `PremasterSecret` of a session, which can be assumed to be unique for each session. Moreover, in most practical cases (where performing the Bleichenbacher attack takes longer than the life time of the TLS session) the attacker would only be able to read encrypted messages after the session has finished, but usually not be able to impersonate the server or to inject adversarially-chosen messages. In our attack against TLS 1.3, the attacker is able to impersonate the server, however, the attacker has to compute a new

[1] `http://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html`

[2] `http://tools.ietf.org/html/draft-tsvwg-quic-protocol-01`

[3] We explain below why this is a very reasonable assumption.

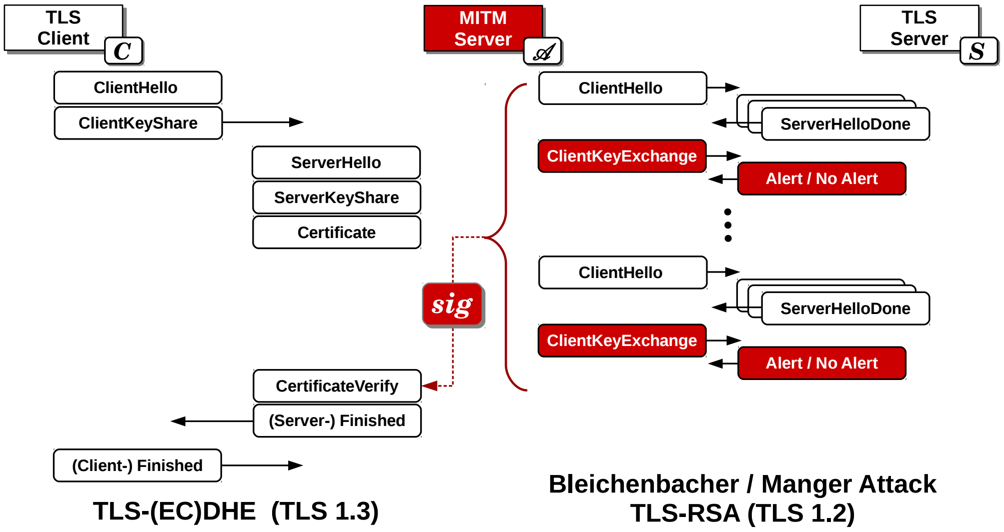

Figure 1: Simplified illustration of the MITM attack against TLS-(EC)DHE-RSA.

signature by mounting a Bleichenbacher-attack for each attacked TLS session, which may be infeasible if this step takes too long.

In contrast, an attacker which obtains a validly signed QUIC SCFG (serialized server config) message, by performing Bleichenbacher's attack *once*, is able to impersonate the server in an *arbitrary number of sessions* and *against an arbitrary number of clients*, until the SCFG message has expired. Note that the expiry date is also chosen by the attacker, thus, this date can be as far ahead in the future as the attacker likes. For the attacker this is *essentially equivalent to obtaining the server's secret key* — by performing only a *single* Bleichenbacher attack.

Thus, in the realistic setting where a web server uses an X.509 certificate for PKCS#1 v1.5 encryption in TLS, and where this same (valid, legitimate) certificate is accepted by a TLS 1.3/QUIC client has a serious but still limited impact on the security of TLS, but a completely devastating impact on the security of QUIC. Note that this works even if the real QUIC server actually uses a completely different certificate.

Cross-Ciphersuite Attacks. At CCS'12, a *cross-ciphersuite* attack for TLS was presented [26], and at CCS'14, a formal model for this type of attack was proposed [5]. Both papers have extended the research on cryptographic protocols to cover more practically relevant aspects. The present paper can be seen as an extension to this line of research, by proposing the first *cross-ciphersuite-family* attack from TLS-RSA to TLS-DHE, the first *cross-protocol-version* attack (TLS 1.2 to TLS 1.3), and even the first *cross-protocol* attack (TLS to QUIC). Moreover, while the success probability of the attack from [26] was fixed to 2^{-40}, the success probability of our attack depends on the environment and may even be completely realistic against QUIC.

Practical evaluation. We conducted experiments to assess the feasibility of this attack. There are no reference implementations of TLS 1.3 yet available, and for QUIC there is only experimental server code available, which unfortunately does not support signed SCFG messages. However, note that the approach extends easily to previous TLS versions, as a cross-ciphersuite attack. Therefore we analyzed a server which implements only TLS version 1.2, but offers two different cipher suites, one from the `TLS_RSA` family which is vulnerable, and one from the `TLS_ECDHE_RSA` family (to mimic TLS 1.3), using the same RSA key for both cipher suites. We consider a client that accepts only `TLS_ECDHE_RSA` cipher suites. This provides essentially the conditions required to analyze the attack principle described above.

In order to simulate a vulnerability of the `TLS_RSA` cipher suite, we patched the OpenSSL 1.0.2 TLS implementation,[4] such that it becomes vulnerable to an adoption of Manger's attack [25] to PKCS#1 v1.5 encryption, which is a very efficient "Bleichenbacher-type" attack. The man-in-the-middle attacker is written in Java 7. We tested the attack with different web browsers (including Microsoft's Internet Explorer 11 on Windows 7, Apple's Safari 7.1.3 on OS X 10.9.5, and Firefox 35 and Google Chrome 39 on Ubuntu Linux 14.04. We also tested the `OpenSSL` TLS client (used, for instance, in machine-to-machine TLS communication) on Ubuntu Linux 14.04.

The experimental results show that the attack can be performed within 30 seconds (for 1024 bit moduli). While this appears very practical, we caution that our analysis is based on a modified OpenSSL server S that provides a very strong, "ideal" oracle that allows one to distinguish valid from invalid ciphertexts. For instance, the recent Bleichenbacher-attacks of Meyer et al. [28] take at least about 20 hours. Thus, these attacks should not (yet) be considered practical.

For QUIC the situation is completely different, because in this case the Bleichenbacher-attack can be executed long before the victim client initiates a session, which makes the attack truly practical even if it requires 20 hours (or more) of precomputation.

The difficulty of preventing this attack. A first obvious solution would be to either deactivate previous TLS versions, or at least the vulnerable cipher suites in these versions. However, the former will usually not be possible, because server operators may want to keep older cipher suites for compatibility with older clients. The latter is not possible without breaking standard-conformance, because the only mandatory-to-implement cipher suites in TLS 1.1 and 1.2 are based on PKCS#1 v1.5-encrypted key transport. Moreover, certain important browsers (in particular legacy browsers) may possibly not implement QUIC. Thus, the need for *backwards compatibility* and *interoperability* in practice makes it impossible to employ these countermeasures.

[4] OpenSSL 1.0.2 (22-Jan-2015), `www.openssl.org`

Moreover, note that the attack is based on the assumption that the server uses *the same* RSA-certificate in TLS 1.3/QUIC as in older TLS versions, and cipher suites with either RSA encryption or RSA signatures. One generic and cryptographically clean approach for preventing this attack is therefore to enforce *key separation*, that is, to use different keys (and thus different certificates) for different cipher suites and protocol versions. While this is *in theory* the cleanest solution, it has many drawbacks that make it *impractical*.

First of all, note that *basic* X.509 RSA certificates do not contain any information for which algorithm (or TLS version or TLS cipher suite) they shall be used. Thus, even if a server uses a different RSA certificate in TLS 1.3 or QUIC than in other protocol versions, a client would not be able to tell whether a given certificate really belongs to version 1.3. Thus, an attacker would be able to use the certificate from the earlier TLS version in a TLS 1.3/QUIC session with the client, which circumvents the key separation intended by the server.

The best practical solution is to use different keys for encryption and signature verification. X.509 certificates contain a *key usage* extension field, which can be used to indicate that a given certificate can be used only for encryption, or only for signing, etc. Thus, a server operator may use a "sign-only" RSA certificate for TLS 1.3/QUIC, and an "encrypt-only" certificate for previous TLS versions. A technical hurdle for realizing key separation in TLS is that there is no obvious way to configure the popular TLS server implementation `OpenSSL` (also used in Apache's `mod_ssl`, for instance) such that different RSA certificates are used for different TLS versions or different cipher suite families. Moreover, the popular web server `nginx` allows one to deploy only a single certificate. This makes it difficult for users to realize key separation.

Finally, note that any solution that requires the server to use multiple different certificates leads to a more complex (and thus more error-prone) server configuration and higher costs, in particular when expensive *extended validation* certificates are used. Many server operators may want to avoid this.

Generality of the attack principle. We note that a similar attack would work for any protocol whose security is based on RSA signatures, under the same preconditions that we require for our attacks on TLS 1.3 and QUIC. Thus, TLS 1.3 and QUIC can be seen as particular case studies, which demonstrate the practical relevance of the more general attack principle. The core idea behind the attack presented in this paper can be seen as a corollary of the attacks presented in [17, 28]. The main novelty is the application to TLS 1.3 and QUIC.

Note that the attack on QUIC is much more efficient than the attack on TLS 1.3, due to the subtle difference that QUIC allows the server to use the same signature for many protocol sessions, while TLS 1.3 requires to compute the signatures over the random nonces chosen by both communicating parties, which requires a fresh signature for each protocol session, which makes the attack less efficient and therefore less practical. Thus, protocols of the latter type can be seen as more "robust" against this type of attacks, which we consider as an interesting insight.

2. RELATED WORK

In 1996, Wagner and Schneier described the first cross-protocol attack on TLS [33] (called "key exchange algorithm rollback attack" in this paper). The authors made the observation that the digital signature over a key exchange message in TLS does not cover the negotiated cipher suite. This enabled an attacker to take a signed key exchange message from a previous `TLS_DHE_RSA` connection, and let a client interpret it as a `TLS_RSA_EXPORT` key exchange message. Wagner and Schneier explain that a TLS client misinterpreting the `TLS_DHE_RSA` parameters as *cryptographically weak* `TLS_RSA_EXPORT` parameters could be vulnerable to a man-in-the-middle attack. However, this attack was described only theoretically and never applied in practice against a real TLS client. The main problem is that the number of parameters used for `TLS_DHE_RSA` and `TLS_RSA_EXPORT` cipher suites differ, which makes the signature invalid. A more precise analysis is given in [26].

At ACM CCS 2012, Mavrogiannopoulos et al. [26] described a cross-protocol attack on TLS which refines the idea of Wagner and Schneier [33]. The authors considered a combination of `TLS_DHE` and `TLS_ECDHE` cipher suites, and showed the possibility that a TLS client accepts a `TLS_ECDHE` key exchange message in a `TLS_DHE` connection. This was used to let the client misinterpret ECDHE parameters as *cryptographically weak* DHE parameters, which in turn may give rise to a man-in-the-middle attack. Because of the strictly-specified structure of key exchange messages for the considered cipher suites, the analysis in [26] showed that an attacker would needed about 2^{40} signed server messages to execute a TLS man-in-the-middle attack with reasonable probability. This makes the attack rather impractical (the authors estimate that executing the attack in their setup would take about 9 years). Nevertheless, both previously known cross-protocol attacks on TLS [33, 26] give interesting insights into the difficulty of secure protocol design in practice.

In comparison, the performance of our attack depends on the availability of an oracle that allows the attacker to distinguish valid from invalid PKCS#1 v1.5 ciphertexts. In an ideal (but currently hypothetical) case, the attack could be performed in less than 30 seconds (see Section 7). In more realistic cases, based on previously published Bleichenbacher-attacks [28, 36], our attack would take several hours. Thus, like previous works [33, 26], the practical impact of our attack on TLS is (currently) rather limited. However, it provides another interesting insight into the difficulty of secure protocol design in practice, in particular on the difficulty of enforcing key separation in practice and its potential effect on the security of protocols. Moreover, note that both previous cross-protocol attacks [33, 26] are based on the fact that the signature in TLS versions prior to 1.3 did not provide a sufficient binding among cryptographic parameters and used algorithms in key exchange messages. This changes with TLS 1.3, where the signature protects not only the cryptographic parameters, but also the negotiated cipher suite. This makes both previous attacks impossible. Note also that resilience against cross-protocol attacks is an explicit goal of TLS 1.3,[5] and the protocol has been designed to protect against known attacks of this type [33, 26].

It is a well-known fact in cryptographic theory that using the same key with both a weak algorithm and a secure algorithm may force the secure algorithm to inherit the weaknesses of the weaker algorithm, and therefore the principle of *key separation* (i.e., using different keys for different algorithms) should be enforced. For example, the fact that the need for backwards compatibility may lead to attacks was also pointed out in [17], which used the fact that a vulnerable implementation of PKCS#1 v1.5 encryption gives rise to a "signing oracle" to attack XML-based Web Services. A different variant was explored in [9] in the context of EMV signatures, but the overall principle was already mentioned in Bleichenbacher's original paper [7]. Our attacks extend this concept to novel protocols of extremely high practical importance, namely TLS 1.3 and QUIC. Moreover, they demonstrate the difficulty of

[5] See `http://www.ietf.org/proceedings/87/slides/slides-87-tls-5.pdf`.

enforcing proper key separation in practice, and the impact of the lack thereof, on cryptographic protocols as important as TLS and QUIC.

3. TRANSPORT LAYER SECURITY

In the TCP/IP reference model, the TLS protocol is located between the transport layer and the application layer. Its main purpose is to protect insecure application protocols like HTTP or IMAP. The first (inofficial) version was developed in 1994 by Netscape, named *Secure Sockets Layer*. In 1999, SSL version 3.1 was officially standardized by the IETF Working Group and renamed to *Transport Layer Security* [10], version 1.0. Since then, two updates of the TLS specification were released, versions 1.1 [11] and 1.2 [12]. Version 1.3 is currently under development [13].

Cipher suites. TLS is a protocol framework that allows communicating parties to choose from a large number of different algorithms for the various cryptographic tasks performed in the protocol (key agreement, authentication, encryption, integrity protection). A *cipher suite* is a concrete selection of algorithms for all required cryptographic tasks. For example, a connection established with the cipher suite TLS_RSA_WITH_AES_128_CBC_SHA uses RSA-PKCS#1 v1.5 public-key encryption [20] to establish a key, and symmetric AES-CBC encryption with 128-bit key and SHA-1-based HMACs. A connection with cipher suite TLS_DHE_RSA_WITH_AES_128_CBC_SHA uses the same symmetric algorithms, but establishes the key from a Diffie-Hellman key exchange with ephemeral exponents[6] and RSA-PKCS#1 v1.5 signatures [20] for authentication.

The TLS RFCs [10, 11, 12] and their extensions [6] specify a large number of different cipher suites. Only the public-key algorithms used in a TLS session will be relevant for our attack. Therefore we will write "TLS_(EC)DHE_RSA" to denote any cipher suite using (elliptic curve) DHE key exchange and RSA signatures, and "TLS_RSA" to denote any cipher suite based on RSA key transport.

3.1 The TLS Handshake up to Version 1.2

At the beginning of each TLS session the *TLS Handshake* protocol is executed, to negotiate a cipher suite and cryptographic keys. In the following, we give a brief overview of the TLS Handshake for all versions up to 1.2, in as much detail as required to explain our attack. Note that the sequence of messages exchanged in the handshake depends on the selected cipher suite. Version 1.3 will have a slightly different handshake, we explain the differences in Section 3.2.

Handshake overview. A TLS handshake (cf. Figure 2) is initiated by a TLS client with a ClientHello message. This message contains information about the TLS version and a list of references to TLS cipher suites proposed by the client.

The server now responds with the messages ServerHello, Certificate, an optional ServerKeyExchange message, and ServerHelloDone. The ServerHello message contains a reference to a cipher suite, selected by the server from the list proposed by the client. The Certificate message contains an X.509 certificate with the server's public key. The ServerKeyExchange message is optional. It is omitted when a TLS_RSA cipher suite is used, but sent when a TLS-(EC)DHE cipher suite is used. We explain its contents below. The ServerHelloDone message indicates the end of this step. The client responds with a ClientKeyExchange, which we also explain below.

Finally, both parties send the ChangeCipherSpec and Finished messages. The former notifies the receiving peer that subsequent TLS messages will use the newly negotiated cipher suite. The Finished message is necessary to protect against certain attacks (see [27]). After the handshake has finished, the peers can start to exchange payload data, which are protected by the negotiated cryptographic algorithms and keys.

Key establishment and server authentication with TLS_DHE_RSA cipher suites. If a TLS_DHE_RSA cipher suite is used, then the ClientKeyExchange message contains the client's contribution g^c to a Diffie-Hellman key exchange. The ServerKeyExchange message contains the server's contribution g^s to the Diffie-Hellman key, along with a digital signature computed with the RSA-PKCS#1 v1.5 signature scheme. The purpose of the signature is to authenticate the server explicitly (in contrast to the implicit authentication of TLS_RSA cipher suites described below). It is computed over the random nonces r_C and r_S contained in the ClientHello and ServerHello messages, the server's Diffie-Hellman share g^s, and some other data whose details are not relevant for our purposes. The established Diffie-Hellman key g^{cs} is called the PremasterSecret.

Key establishment and server authentication with TLS_RSA cipher suites. If a TLS_RSA cipher suite is used, then the client selects a random PremasterSecret and encrypts it with the RSA-PKCS#1 v1.5 encryption scheme, under the public key contained in the server's certificate. Then it transmits the resulting ciphertext in the ClientKeyExchange message to the server. The server obtains the PremasterSecret by decrypting the ciphertext.

REMARK 1. *The correct handling of decryption errors in this step is of paramount importance for the security of TLS_RSA cipher suites. An attacker which is able to distinguish "valid" from "invalid" RSA-PKCS#1 v1.5 ciphertexts may apply a Bleichenbacher-type attack to break the security of TLS. Thus, the server must not send any error messages if decryption fails. In general it is very difficult to implement this step securely, as even tiny timing differences or other side channels may lead to practical attacks [28, 36].*

Note that there is no explicit server authentication. The server authenticates implicitly, by being able to compute the Finished message correctly. This message depends on the PremasterSecret, thus, the server must have been able to decrypt the ciphertext contained in the ClientKeyExchange message.

On client authentication via TLS. Note that we have described only server-authentication. It is in principle also possible to authenticate clients cryptographically in the TLS handshake, however, this would require *client certificates*. If an application requires client-authentication, then it is much more common to realize this by running an additional protocol over the established TLS channel. For instance, by transmitting a password. TLS is most commonly used with *server-only* authentication, therefore we focus on this setting. However, we stress that our attacks would apply identically to TLS deployments with client certificates.

Derivation of symmetric cryptographic keys. All further handshake messages and all secrets of the TLS session, including the MasterSecret and all encryption and MAC keys, are derived from the PremasterSecret and other public values. Thus, an attacker that is able to compute the PremasterSecret is also able to compute all cryptographic keys of the session.

[6] That is, both communicating partners choose random exponents for each execution of the Diffie-Hellman protocol within TLS. Alternatively, there exist TLS_DH cipher suites, where the server uses a static exponent.

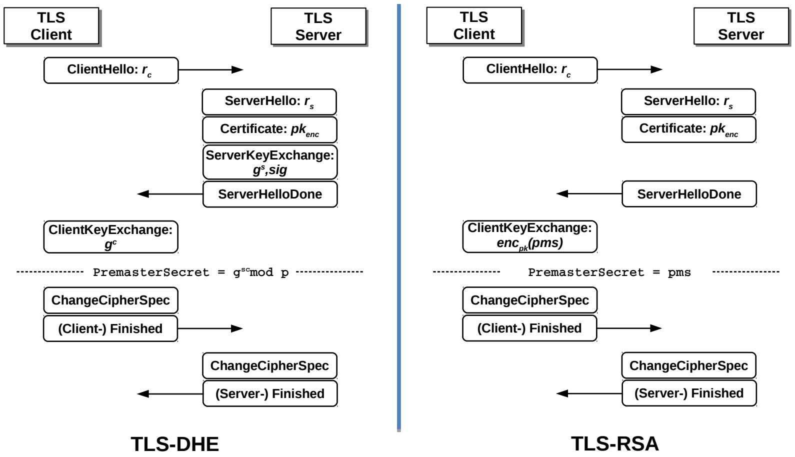

Figure 2: Structure of the SSL/TLS Handshake protocol for `TLS_DHE_RSA` and `TLS_RSA` cipher suites and TLS versions up to 1.2.

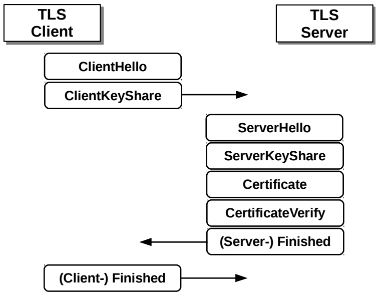

Figure 3: Messages of the TLS 1.3 Handshake.

Mandatory cipher suites. For interoperability reasons, each TLS version specifies cipher suites that are *mandatory* to implement:

- TLS 1.0: `TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA`
- TLS 1.1: `TLS_RSA_WITH_3DES_EDE_CBC_SHA`
- TLS 1.2: `TLS_RSA_WITH_AES_128_CBC_SHA`

Note that TLS versions 1.1 and 1.2 mandate a cipher suite from the `TLS_RSA` family, while version 1.0 mandates a `TLS_DHE` cipher suite, but with DSS signatures instead of RSA.

3.2 Differences in the TLS 1.3 Handshake

TLS version 1.3 [13] is currently under development. One goal of this standard is to improve security of TLS. To this end, obsolete and non-authenticated algorithms are removed, to enforce usage of secure algorithms. In particular, the IETF Working Group decided to remove support of key exchange algorithms based on RSA-encrypted key transport and static-exponent Diffie-Hellman.

In addition to these changes in cryptographic algorithms, TLS 1.3 modifies the TLS Handshake protocol. The server public key parameters for `TLS_(EC)DHE` cipher suites are exchanged in new `ServerKeyShare` messages. These messages are not authenticated directly. Instead, the server sends the public key parameters unauthenticated, followed by further handshake messages. The authentication process is performed with a `CertificateVerify` message which is sent directly before the server `Finished` message. The `CertificateVerify` message includes a PKCS#1 v1.5 RSA signature over a hash of all the previous messages.

A mandatory cipher suite for TLS 1.3 is not yet defined in the current IETF draft [13].

4. THE QUIC PROTOCOL

In the sequel we give a high-level description of the QUIC protocol, in as much detail as required to follow the description of our attacks. A full description is out of scope of this paper. We refer to [32, 8], and the documents referenced in [31] for details. See also [14, 23] for formal security analyses of QUIC.

Slightly simplifying the description contained in [32, pp.18 ff] and [8] (in particular ignoring all countermeasures against potential denial of service attacks), a QUIC connection consists of two phases, the *connect phase* and the *repeat phase* (see Figure 4).

Connect phase. This phase is executed whenever a client connects to a server for the first time. It takes usually one, sometimes even two round-trip times (RTT), and is therefore avoided whenever possible. Its main purpose is to perform authentication between client and server (in particular, the server's X.509 certificate is transmitted to the client in this phase), and to establish a state from which future session keys can be derived in the zero-RTT *repeat phase* described below.

Most importantly for our work, in this phase the server transmits a *serialized server config* (SCFG) message, which contains information about supported elliptic curves and encryption algorithms, an elliptic curve Diffie-Hellman share g^S as the server's contribution to a Diffie-Hellman key exchange, and a 64-bit *expiry time* which indicates the expiration date of the given parameters. After the SCFG message has expired, a new connect phase must be performed. The SCFG message is digitally signed with the secret key corresponding to the server's X.509 certificate.

If the server uses an RSA-certificate, which is probably the most common scenario, then the RSA-PSS signature algorithm is used to compute the signature. Please note that only input from the server is signed, in contrast to TLS, where the client nonce (which may be assumed to be unique for each session) is signed, too.

Repeat Phase. This phase is executed immediately after the *connect phase*, and whenever the client later connects to the server (provided that the time stamp in the stored SCFG message has not expired and that the server still uses the same certificate).

Most importantly for us, this message contains the client's contribution g^C to the elliptic curve Diffie-Hellman key shared between client and server. Moreover, this message may also contain encrypted payload, encrypted with a shared key derived from the mutual Diffie-Hellman key g^{CS}.

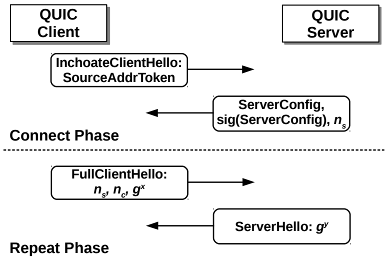
QUIC Client
QUIC Server
InchoateClientHello: SourceAddrToken
ServerConfig, sig(ServerConfig), n_s
Connect Phase
FullClientHello: n_s, n_c, g^x
ServerHello: g^y
Repeat Phase

Figure 4: Two phases in the QUIC protocol.

Analysis. We make the following two observations, which are crucial for the practicality of our second attack.

1. The repeat-phase does not perform any additional server authentication. The server is authenticated explicitly in the connect-phase, by the signature over the SCFG message containing g^S, and only implicitly in the repeat phase by being able to compute the shared key derived from g^{CS}.

2. The signed SCFG message transmitted in the connect-phase is *independent* of a client's connection request. Therefore the SCFG message can be pre-computed by a server (with an appropriate expiration date) or — as we will show — by an attacker which is able to compute forged signatures for arbitrary messages.

5. RSA ENCRYPTION

For completeness, we describe PKCS#1 v1.5 encryption and signatures [20] in this section, and give high-level descriptions of the attacks of Bleichenbacher [7] and Manger [25]. The reader may safely skip these details, keeping only the following in mind:

- For a given RSA public key (N, e) with secret key d, both attacks enable an attacker to compute the "textbook" RSA decryption (resp. RSA signature) function $m \mapsto m^d \bmod N$ without knowing d for all values $m \in \mathbb{Z}_N$.
- This is sufficient to compute an RSA-PSS or an RSA-PKCS#1 v1.5 signature σ for any message M, such that σ is valid with respect to M and (N, e).

Note also that RFC 2313 [20], which specifies PKCS#1 v1.5 encryption and signatures, is obsolete [21, 19]. However, TLS still uses this version for backwards compatibility reasons.

5.1 RSA-Signatures and RSA-Encryption according to PKCS#1 v1.5

In the sequel let (N, e) be an RSA public key, with corresponding secret key d. We denote with ℓ the byte-length of N, thus, we have $2^{8(\ell-1)} < N < 2^{8\ell}$.

Digital signatures. Let $H : \{0,1\}^* \to \{0,1\}^{8\ell_H}$ be a cryptographic hash function (e.g. SHA-1) with ℓ_H-byte output length. A digital signature over message M according to RSA-PKCS#1 v1.5 [20], as used in all TLS versions, is computed in three steps.

1. Compute the hash value $h := H(M)$.
2. Compute a padded message
$$m := \texttt{0x01}||\texttt{0xFF}||\ldots||\texttt{0xFF}||\texttt{0x00}||\mathsf{ASN.1}||h$$
where $\mathsf{ASN.1}$ is a 15-byte string which identifies the hash function H in ASN.1 format.
3. Interpret m as an integer such that $0 < m < N$ and compute the signature as $\sigma := m^d \bmod N$.

A digital signature over message M according to the RSA-PSS signature scheme [19], as used in QUIC, is computed identically, except that instead of the simple padding in Step 2 of the above algorithm a more complex, probabilistic padding scheme is used to compute the padded message m.

Public-key encryption. The basic idea of PKCS#1 v1.5 encryption is to take a message k (a bit string), concatenate this message with a random padding string PS, and then apply the RSA encryption function $m \mapsto m^e \bmod N$. More precisely, a message k of byte-length $|k| \leq \ell - 11$ is encrypted as follows.

1. Choose a random padding string PS of byte-length $\ell - 3 - |k|$, such that PS contains no zero byte. The byte-length $|PS|$ of PS must be at least $|PS| \geq 8$.
2. Set $m := \texttt{0x00}||\texttt{0x02}||PS||\texttt{0x00}||k$.
3. Interpret m as an integer such that $0 < m < N$ and compute the ciphertext as $c := m^e \bmod N$.

The decryption algorithm computes $m' = c^d \bmod N$ and interprets integer m' as a bit string. It tests whether m' has the correct format, i.e. whether m' can be parsed as $m' = \texttt{0x00}||\texttt{0x02}||PS||\texttt{0x00}||k$, where PS consists of at least 8 non-zero bytes. If this holds, then it returns k, otherwise it rejects the ciphertext.

5.2 Bleichenbacher's Attack

In this section, we recall the well-known attack of Bleichenbacher [7] on RSA-PKCS#1 v1.5 encryption [20]. Essentially, Bleichenbacher's attack allows one to compute the "textbook" RSA-decryption (resp. RSA-signing) function $m \mapsto m^d \bmod N$ without knowing the secret exponent d or (equivalently) the factorization of N, by exploiting the availability of a *ciphertext validity oracle*. This gives rise to attacks on cryptosystems relying on the one-wayness of the RSA function, like RSA-PKCS#1 v1.5 encryption and signatures.

Prerequisites. Bleichenbacher's attack assumes an oracle $\mathcal{O}_{\mathsf{BI}}$ which tells whether a given ciphertext is valid (that is, PKCS#1 v1.5 conformant) with respect to the target public key (N, e). This oracle takes as input a ciphertext c and responds as follows.

$$\mathcal{O}_{\mathsf{BI}}(c) = \begin{cases} 1 \text{ if } c \text{ is valid w.r.t. PKCS\#1 v1.5 and } (N, e), \\ 0 \text{ otherwise.} \end{cases}$$

The oracle abstracts the availability of, for instance, a web server responding with appropriate error messages. We note that this oracle does not need to be "perfect". That is, Bleichenbacher's algorithm works even if the oracle occasionally returns false-negatives, which occur if $\mathcal{O}_{\mathsf{BI}}(c)$ returns 0 but c is a valid PKCS#1 v1.5 ciphertext.

High-level attack description. We give only a high-level description of the attack, and refer to the original paper [7] for details. Suppose a PKCS#1 v1.5 conformant ciphertext $c = m^e \bmod N$ is given. Thus, $m = c^d \bmod N$ lies in the interval $[2B, 3B)$, where $B = 2^{8(\ell-2)}$. Bleichenbacher's algorithm proceeds as follows. It chooses a small integer s, computes

$$c' = (c \cdot s^e) \bmod N = (ms)^e \bmod N,$$

and queries the oracle with c'. If $\mathcal{O}_{\mathsf{BI}}(c') = 1$, then the algorithm learns that $2B \leq ms - rN < 3B$ for some small integer r which is equivalent to

$$\frac{2B + rN}{s} \leq m < \frac{3B + rN}{s}.$$

By iteratively choosing new s, the adversary reduces the number of possible values of m, until only one is left.

Attack efficiency. For a 1024-bit modulus and a random ciphertext, the original analysis in [7] shows that the attack requires about one million oracle queries to recover a plaintext. Therefore, Bleichenbacher's attack became also known as the "Million Message Attack". Recent improvements in cryptanalysis [4] show, however, that this number can be significantly improved. In particular, in certain (realistic) scenarios the improved attack of [4] performs only about 3800 oracle queries, depending on which ciphertext validity checks are performed by the oracle.

5.3 Manger's Attack

Subsequent to Bleichenbacher, Manger [25] described an attack on RSA-PKCS#1 v2.0 encryption [21] (aka. RSA-OAEP). Like Bleichenbacher's attack, Manger's attack allows one to compute the RSA-decryption (resp. RSA-signing) function $m \mapsto m^d \bmod N$ without knowing the secret exponent d or (equivalently) the factorization of N, by exploiting the availability of a *ciphertext validity oracle*, but under different prerequisites and with better efficiency.

We note that Manger's attack can be easily adopted to RSA-PKCS#1 v1.5, provided that an oracle is given which checks only whether the first byte is zero.

Prerequisites. Manger's attack assumes an oracle $\mathcal{O}_{\mathsf{Ma}}$ which tells whether for a given ciphertext c, the value $c^d \bmod N$ (interpreted as a byte array) begins with 0x00. Thus, this oracle takes as input a ciphertext c and responds as follows.

$$\mathcal{O}_{\mathsf{Ma}}(c) = \begin{cases} 1 \text{ if } c^d \bmod N \text{ begins with } \texttt{0x00}, \\ 0 \text{ otherwise.} \end{cases}$$

Let $B = 2^{8(\ell-1)}$, so that any number in $\mathbb{Z}_N$ less than B will start with a 0x00-byte. Thus, the oracle tells for a given ciphertext c whether $m = c^d \bmod N$ lies in the interval $[0, B-1]$ (if the oracle outputs 1) or in $[B, N-1]$ (if the oracle outputs 0).

We need to assume that this oracle is "perfect", in the sense that it always responds correctly. It is not able to tolerate false-positives or false-negatives.

High-level attack description. Again, we give only a high-level description of the attack, and refer to the original paper [25] for details. Suppose $c = m^e \bmod N$ is given, with $m < B$. Manger's algorithm proceeds very similarly to Bleichenbacher's algorithm, by choosing a small integer s, computing $c' = (c \cdot s^e) \bmod N = (ms)^e \bmod N$, and querying the oracle with c'.

The main difference to Bleichenbacher's algorithm is that Manger's approach makes essential use of the "perfectness" of the oracle, which allows one to choose values s in a more sophisticated way. That is, the information whether $m \cdot s \bmod N$ lies in $[0, B-1]$ or not, which is provided by $\mathcal{O}_{\mathsf{Ma}}$, reveals (almost) one bit of information about m.

Attack efficiency. For a 1024-bit modulus and a random ciphertext, the original analysis in [25] shows that the attack requires only about 1100 oracle queries to invert the RSA function (note that this is close to optimal). However, in contrast to Bleichenbacher's attack, Manger's attack needs a "perfect" oracle which always responds correctly. It is not able to tolerate false-positives or false-negatives.

6. ATTACKS ON TLS 1.3 AND QUIC

We consider a victim client C that establishes a TLS/QUIC connection to a web server S. The web server uses an X.509 certificate containing an RSA key pair with public key (N, e) and secret key d for the TLS connection. The certificate is digitally signed by a *certification authority* (CA) trusted by C. Of course, we neither assume that the attacker is able to corrupt the CA (or any other CA trusted by the client), nor that the client does not verify certificates properly [15], as otherwise a man-in-the-middle attack would be trivial.

The goal of our attacker is to impersonate the server S towards C, to be able to mount a man-in-the-middle attack that enables it to read and modify the TLS-encrypted data exchanged between C and S. For clarity, let us summarize and discuss the assumptions that we make.

Standard network attacker model. We work in the standard network attacker model. Essentially, we assume that the attacker's host is located on the network path between C and S. Even in settings where the attacker does not control the network, this can often be easily realized by spoofing attacks, which are usually very simple when the attacker is in the same local network as the victim. We also assume that the attacker is able to establish connections to S. For example, S may be a publicly available web server on the the Internet.

Server provides at least one vulnerable `TLS_RSA` ***cipher suite.*** We assume that the server offers at least one cipher suite from the `TLS_RSA` family, such that the implementation of RSA-encrypted key transport in this cipher suite is vulnerable to an attack that allows an attacker to compute the function $m \mapsto m^d \bmod N$ for any value $m \in \mathbb{Z}_N$. This may, for instance, be Bleichenbacher's or Manger's attack (cf. Sections 5.2 and 5.3).

Note that it is very common for a server to offer multiple TLS versions and multiple cipher suites, for compatibility reasons. Moreover, a web server that offers TLS Versions 1.1 or 1.2 *must* offer a cipher suite from the `TLS_RSA` family, as the mandatory to implement [11, 12] cipher suites for these versions belong to this family. As explained in the introduction, this paper is written under the hypothesis that attacks like Bleichenbacher's and Manger's on PKCS#1 v1.5 encryption remain a non-negligible threat.

Server uses an RSA-certificate that allows for signing. We assume that the web server uses a certificate which may be used for RSA signatures. Note that it is in principle possible to create X.509 certificates which may only be used in certain applications. That is, an X.509 certificate may contain a *key extension* field, which specifies that the certificate can only be used for encryption, or only for digital signatures.

A web server may offer different cipher suites, where some cipher suites use RSA signatures (e.g., any cipher suite of the `TLS_DHE_RSA` family), and some others use RSA encryption (e.g., any cipher suite of the `TLS_RSA` family). This is a very common scenario. Such a web server would in principle be able to use different certificates for different cipher suites. However, we argue that this is extremely uncommon. First, it seems not even possible (to our best knowledge) to configure the most popular TLS implementation `OpenSSL`[7] in a way such that different RSA certificates are used for different cipher suites or protocol versions. Similarly, the popular `nginx` web server[8] allows one to use only a *single* server certificate [30]. Thus, any such server which offers at least one `TLS_RSA` cipher suite and one `TLS_(EC)DHE_RSA` cipher

[7] `https://www.openssl.org`

[8] `http://nginx.org`

suite will have to use an RSA-certificate that allows for *both* signing and encryption. Second, certificates signed by commercial CAs are costly, in particular *extended validation* certificates, which cost several hundred US dollars *per year*. Therefore many server operators might not want to buy different certificates for different cipher suites and TLS protocol versions.

Client accepts RSA signatures. For both attacks on TLS 1.3 and QUIC we assume that the client accepts RSA server certificates (for RSA signatures in TLS 1.3 and QUIC). Note that the majority of certificates on the Internet is RSA-based, thus, any client not accepting RSA-certificates at all would not be able to establish TLS connections to a large number of web sites. Moreover, `TLS_(EC)DHE_RSA` cipher suites are *explicitly recommended* by security experts [30] for security-critical applications.

6.1 The Attack on TLS 1.3

For concreteness, let us consider a server S that supports TLS versions 1.2 and 1.3, and a client that accepts only TLS 1.3 connections. The attack generalizes easily to other settings that satisfy the above conditions. We write TLS-RSA to denote an arbitrary cipher suite from the `TLS_RSA` family offered by the TLS 1.2 implementation, and TLS-DHE-RSA to denote an arbitrary cipher suite from the `TLS_DHE_RSA` family offered by the TLS 1.3 implementation. The attack proceeds as follows (cf. Figure 1).

1. The client C sends the TLS 1.3 messages `ClientHello` and `ClientKeyShare`, which contain (among other values) a list of cipher suites accepted by the client C.

2. The attacker $\mathcal{A}$ intercepts these messages. He selects an TLS-DHE-RSA cipher suite. Then it responds to C with a `ServerHello` message, which contains the selected cipher suite. Then it chooses a random Diffie-Hellman exponent $a \overset{\$}{\leftarrow} \mathbb{Z}_{|G|}$ and responds with a `ServerKeyShare`-message containing g^a. Note that the attacker knows the Diffie-Hellman exponent a.

3. $\mathcal{A}$ now retrieves the server's RSA certificate by sending a `ClientHello` message (for an arbitrary TLS version) to S. The server responds with a corresponding `ServerHello` message and its certificate. The attacker embeds the retrieved certificate in a `Certificate` message and forwards it to C.

4. In order to finish the establishment of a TLS 1.3 session with C, the attacker now has to compute the `CertificateVerify` message. This message must contain a signature over the transcript M of all previously exchanged messages. The signature must be valid with respect to M and the public key (N, e) contained in the certificate of S. To this end, the attacker first computes the PKCS#1 v1.5-signature encoding m of M (see Section 5.1). Let m denote the result. Then it computes the signature $\sigma = m^d \bmod N$, using that the vulnerability of the server allows for computing the function $m \mapsto m^d \bmod N$ for all $m \in \mathbb{Z}_N$. This is sufficient to compute a PKCS#1-v1.5 signature for m that is valid with respect to the server's public key.

5. Using its knowledge of the exponent a, the attacker is now able to compute all further handshake messages and the PremasterSecret, and thus all other secrets used in the TLS connection with C. Therefore it is able to finish the TLS Handshake with C. This establishes a rogue TLS 1.3 connection between C and $\mathcal{A}$, where C believes it communicates with S.

Note that the attack described above is an "online" attack. That is, the MITM attacker is not able to compute the forged signature *before* it receives the `ClientHello` message from the client C. Note also that the execution of the signature forgery computed by the MITM attacker takes some time, as even efficient variants of Bleichenbacher's attack (like Manger's attack) require at least a few thousand server requests. The TLS client has to wait and keep the TLS connection open while the MITM attacker performs these computations. Therefore the efficiency and practicability of the attack depend mainly on the time needed to execute this step. A client may not be willing to wait for a very long time for the `CertificateVerify`-message, after the `ClientKeyShare`-message has been sent. We analyze this in Section 7.

6.2 The Attack on QUIC

The description of the attack on QUIC is much simpler than the attack on TLS 1.3. As already explained in Section 4, we only have to explain how an attacker is able to obtain a validly-signed *serialized server config* (SCFG) message containing

- an elliptic curve Diffie-Hellman share g^A such that the exponent A is known to the attacker (for example, the attacker may choose g^A herself), and

- a time stamp which lets the SCFG message expire at a suitable point in time in the (far) future.

Note that this allows the attacker to impersonate a server an arbitrary number of times (until the SCFG message expires, however, the expiration time can be chosen by the attacker) and against an arbitrary number of different clients. Thus, from the attacker's point of view knowing the SCFG message is essentially equivalent to knowing the server's secret key.

Assuming that the vulnerable TLS server uses an RSA-based certificate that allows for message signing (that is, the *key extension* field does not limit the certificate to encryption-only), the attacker is able to use Bleichenbacher's or Manger's attack against the TLS server to compute a valid signature, exactly as in the attack on TLS 1.3 described above. The only difference is that now the attacker computes an RSA-PSS signature for a SCFG message, which contains a Diffie-Hellman share and an expiration date of the attacker's choice.

Note that the attack works even if the QUIC server uses a different X.509 certificate, because the client is not able to tell whether a given X.509 certificate "belongs to" the TLS server or the QUIC server. The attacker would simply take the certificate from the TLS server, and present it to the attacked client as the certificate for the QUIC protocol. This is possible because the X.509 certificate does not contain any information for which protocol this certificate should be used.

Recall that SCFG messages are independent of any connection request by a client, which allow one to pre-compute SCFG messages prior to the connection attempt of the client. Therefore, even if mounting the Bleichenbacher/Manger attack against the vulnerable server takes a long time (say, 10 days or more, which is far beyond the figures provided by recent examples of Bleichenbacher attacks in practice [4, 28, 36]), the attacker will eventually obtain a validly signed SCFG message.

In a sense, this shows that including client nonces in signatures, as done in TLS, strengthens a protocol against this type of offline attacks.

7. PRACTICAL EVALUATION

7.1 Attacks with "Perfect" Oracle

We will not be able to evaluate the feasibility of the attack on TLS 1.3 directly, because this TLS version is currently in development, and reference implementations are not yet available. However, note that the attacker $\mathcal{A}$ in the attack from Section 6 essentially implements the full TLS 1.3 protocol, with the only exception that it is not in possession of the secret key corresponding to the public key in the server's RSA certificate. Instead, it uses the vulnerability of the server S to obtain an "RSA signing oracle", which essentially computes signatures for messages of the attacker's choice. This is sufficient for $\mathcal{A}$ to impersonate S against C.

This approach extends easily to other TLS versions. Therefore we will evaluate the attack with respect to a server that implements only TLS 1.2, but offers two different cipher suites, one from the `TLS_RSA` family and one from the `TLS_DHE_RSA` family, and a client which only accepts `TLS_DHE_RSA` cipher suites. This mimics the situation described in Section 6 very closely. Even though there are some minor differences between the ordering of the messages and the signed values (note that in TLS 1.2 only a subset of all previously exchanged data is signed), the principle of the attack is exactly the same. The TLS version used by the client has no noticeable effect on the practicability of the attack.

Test Setup. In order to assess the practicability of our attack, we implemented a malicious MITM server $\mathcal{A}$ and tested the attack against different TLS clients C. The MITM server performs a Manger attack [25] (adopted to PKCS#1 v1.5 encryption) against a TLS-RSA server S, which implements a patched OpenSSL server. For simplicity, S was run on the same system as the MITM server (note that this is a realistic assumption for cloud computing environments [29]).

The MITM server and the server S run on a machine with Ubuntu 14.04, with two 2.2 GHz processors and Java 7 (version 1.7.0_75). For the clients, we used different machines with different systems, depending on the tested TLS client software. We tested Google Chrome 39, Mozilla Firefox 35, and `OpenSSL` on Ubuntu 14.04, Safari 7.1.3 on OSX 10.9.5, and Microsoft Internet Explorer on Windows 7.

Experimental results. The approximate time required to compute one forged signature and the number of oracle queries is given in Table 1, for different RSA key-lengths. Note that if an 1024-bit key is used, then the time to compute a forged RSA signature is below 30 seconds, but increases with larger key sizes.

RSA mod. length	# of queries	Duration [sec]
1024	1100	28
2048	2120	66
4096	4200	250

Table 1: Number of queries and time needed to execute Manger's attack against the patched OpenSSL server to create a forged PKCS#1 signature.

According to a recent study by Indutny [16] from April 2015, there are about 34% of Alexa top one million web sites using 1024-bit RSA keys. 63% of the analyzed web sites use 2048-bit keys and 2% use 4096-bit keys. About 1760 web sites use 512-bit RSA keys.

Vulnerability of web browsers. As illustrated in Table 1, the duration of the attack depends on the size of the RSA modulus. Recall that our attack is an "online" attack. That is, the MITM attacker can only begin to compute the forged signature after it has received the `ClientHello` message from the client. This may make the attack impossible, if the TLS client raises a timeout and aborts the establishment of the TLS session before the MITM attacker has computed the signature (and thus is able to respond to the client).

We have analyzed this timeout for different popular web browsers. To this end, we equipped our MITM server with a custom TLS stack, which takes an additional "delay parameter". When receiving a `ClientHello` message, the server responds immediately with the `ServerHello` and the `Certificate` message. The `ServerKeyExchange` message is delayed by the configured time period. In case the delay does not raise a timeout at the client, we increase the delay and reinitialize the connection establishment, until the timeout of the considered web browser is determined. The results of this analysis are depicted in Table 2. For example, Google Chrome 39 strictly closes the connection after 30 seconds and displays a *This webpage is not available* message, which makes our attack feasible for key sizes up to 1024 bits, but impossible for 2048 bit and beyond. In contrast, Mozilla Firefox 35 allows a timeout of 600 seconds, and thus enabled the attack for all considered key sizes.

Please note that for Mozilla Firefox, it is also possible to keep the connection alive indefinitely, by using a technique by Adrian et al. [2]. To this end, the authors used TLS warning alerts.

TLS Client	Connection Open [sec]
Google Chrome 39 (Ubuntu 14.04)	30
Microsoft IE 11 (Windows 7)	450
Mozilla Firefox 35 (Ubuntu 14.04)	600
Safari 7.1.3 (OSX 10.9.5)	450

Table 2: Maximum possible time period for keeping the connection between our MITM server and web browsers alive.

We stress that we have conducted these experiments with an "ideal" oracle that allows the MITM attacker to use the very efficient algorithm of Manger, which requires only 1100 "oracle queries" to compute a forged signature. In practice, a weaker oracle may be given. For instance, typical Bleichenbacher attacks take between about 4000 and a few millions of queries (for 1024 bit RSA keys) [4], depending on the strength of the given oracle. Consequently, the attack duration would be longer. However, we also stress again that we do not claim that the attack is yet practical, our experiments should mainly demonstrate that such attacks are *in principle* possible against TLS 1.3.

Attacking TLS clients beyond web browsers. Note that the time required to execute the MITM attack incurs a noticeable delay until the web browser is able to display the web page. While it may be realistic to assume that a user waits for, say, 30 seconds until a web page has loaded (which is long, but not entirely unrealistic for slow connections, like for instance in public WiFi networks at airports), it becomes increasingly unrealistic with increasing duration of the attack (unless one makes additional assumptions, like that the user works in a different browser tab until the web page has loaded).

However, note that there are also applications of TLS where no human user is involved. TLS-protected machine-to-machine communication is, for instance, common for Web Services. In such applications, we do not have as strict constraints on the running time of the attack as in settings involving human users, because a client machine may allow for a more generous timeout.

Therefore we also analyzed `OpenSSL` on Ubuntu 14.04 as a TLS *client*. It turns out that even after 7700 seconds no timeout of the TLS connection occurs, which could allow for more realistic attacks in settings with machine-to-machine communication that do not involve human users directly.

Avoiding TCP timeouts. Note that the timeout of 7700 seconds in `OpenSSL` is larger than the TCP connection timeout, which defaults to 7200 s in most operation systems (Linux, OS X, and Windows), including the system used for our tests. We were able to increase the TCP connection timeout with the following trick. After receiving the `ClientHello` message, the MITM attacker transmitted the response (in particular the rather large `Certificate` message) *byte-by-byte* over the TCP connection, with a short delay after each byte. This trick avoided the TCP connection timeout.

7.2 Attacks with "Imperfect" Oracle

The consideration of attacks with a "perfect" oracle in the previous section is of course idealized. Even though it is not impossible to find such oracles in practice (see [4] for an example), it is relatively unlikely that such oracles are found very often. The analysis in Section 7.1, Table 1, shows that an attack duration of 30 seconds per 1000 server requests is a reasonable estimate for Bleichenbacher-like attacks in our setting. Using this result, we can now estimate the attack duration in cases where a weaker, "imperfect" Bleichenbacher oracle is given.

Bardou et al. [4] describe an optimized variant of Bleichenbacher's algorithm, and analyze this algorithm with different "imperfect" oracles. The choice of oracles considered in [4] is motivated by examples of practical Bleichenbacher-oracles found in practice. The considered class of oracles starts from a "TTT"-oracle, which performs only few PKCS#1 v1.5 consistency checks and therefore returns 1 on all plaintexts beginning with 0x00||0x02. Such an oracle allows for very efficient Bleichenbacher-attacks, essentially because it is relatively likely that the oracle returns 1 when given a random RSA-ciphertext (not necessarily correctly padded). Such an oracle was found on RSA SecurID and Siemens CardOS smartcards [4], for instance. The most restrictive oracle considered by Bardou et al. is the "FFF"-oracle, which checks PKCS#1 v1.5 consistency very thoroughly, by testing all padding fields and the length of the plaintext for correctness. This makes Bleichenbacher-like attacks less efficient, because it takes rather long until Bleichenbacher's algorithm finds a ciphertext which is accepted by the oracle. This is the type of oracle was found and exploited by Klíma et al. [22] in old TLS versions. Bardou et al. furthermore describe many intermediate oracles, dubbed "TFT", "FFT", and so on, which perform or omit different checks on PKCS#1 v1.5 plaintext padding, we refer to [4] for details.

Bardou et al. compute the number of attack queries for different oracle types. However, their assumption is that the attacked message is decrypted to a message starting with 0x00||0x02, which speeds up the attack. In our case, we have to perform RSA secret key operation on arbitrary messages. Thus, our first step is to find a message starting with 0x00||0x02 (called a *blinding step* in the original paper of Bleichenbacher [7]). The performance of this step depends on the oracle strength. For example, it takes about 2^{15} additional queries to find a valid message, when the oracle validates only the first two bytes 0x00||0x02 ("TTT" oracle).

In Table 3 we give the estimated duration of the attack in our setting with the oracles from [4]. The number of oracle queries is computed as a sum of the queries in the blinding step and queries to perform the attack by Bardou et al. [4].

On attacking TLS 1.3. Considering attacks on TLS 1.3, we observe that none of the durations lies within the "session keep-alive" time of the tested browsers (cf. Table 2). However, the more interesting case for attacks where the duration is very long is machine-to-machine communication. Note that for OpenSSL it holds that all timing results are within the timing range, except for the case of the FFF oracle.

Oracle type	#Queries	Duration [sec]
TTT	36,536	1,097
TFT	37,796	1,134
FFT	59,388	1,782
FFF	24,228,692	726,861

Table 3: Estimated duration of attacks for 1024-bit RSA keys with the imperfect Bleichenbacher oracles found in [4], based on the median number of queries of the optimized Bleichenbacher algorithm from [4].

On attacking QUIC. Considering attacks on QUIC, note that the time required to perform one full Bleichenbacher attack ranges from about 18 minutes (for TTT and TFT oracles) up to about 202 hours (less than 8.5 days) in case of the most restrictive FFF oracle. Given that in case of QUIC the attacker is able to perform these computations before the actual attack in a "pre-computation", we consider these figures as fully practical, even for the most restrictive FFF oracle.

8. ACKNOWLEDGEMENTS

We would like to thank Adam Langley for his advice on the limitations of the QUIC reference implementation and further comments, Fedor Indutny for providing us an early version of his certificate collection, the reviewers of USENIX Security 2015 for suggesting to analyze the applicability of our attacks to QUIC, and the reviewers of ACM CCS 2015 for their helpful comments.

9. REFERENCES

[1] SSL Pulse. Survey of the SSL Implementation of the Most Popular Web Sites, April 2015. https://www.trustworthyinternet.org/ssl-pulse.

[2] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, Paul Zimmermann. Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. May 2015. https://WeakDH.org

[3] Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors. *ACM CCS 14: 21st Conference on Computer and Communications Security*. ACM Press, November 2014.

[4] Romain Bardou, Riccardo Focardi, Yusuke Kawamoto, Lorenzo Simionato, Graham Steel, and Joe-Kai Tsay. Efficient padding oracle attacks on cryptographic hardware. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 608–625. Springer, August 2012.

[5] Florian Bergsma, Benjamin Dowling, Florian Kohlar, Jörg Schwenk, and Douglas Stebila. Multi-ciphersuite security of the secure shell (SSH) protocol. In Ahn et al. [3], pages 369–381.

[6] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). RFC 4492 (Informational), May 2006. Updated by RFC 5246.

[7] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1.

In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer, August 1998.

[8] Wan-Teh Chang and Adam Langley. QUIC crypto, 2013. `https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/edit?pli=1`.

[9] Jean Paul Degabriele, Anja Lehmann, Kenneth G. Paterson, Nigel P. Smart, and Mario Strefler. On the joint security of encryption and signature in EMV. In Orr Dunkelman, editor, *Topics in Cryptology – CT-RSA 2012*, volume 7178 of *Lecture Notes in Computer Science*, pages 116–135. Springer, February / March 2012.

[10] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999. Obsoleted by RFC 4346, updated by RFCs 3546, 5746.

[11] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), April 2006. Obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746.

[12] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFC 5746.

[13] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. draft-ietf-tls-tls13-07, July 2015.

[14] Marc Fischlin and Felix Günther. Multi-stage key exchange and the case of google's QUIC protocol. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1193–1204. ACM, 2014.

[15] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software. In *ACM Conference on Computer and Communications Security*, 2012.

[16] Fedor Indutny. Rsa certificate sizes, April 2015. `http://indutny.github.io/collect-certs`.

[17] Tibor Jager, Kenneth G. Paterson, and Juraj Somorovsky. One bad apple: Backwards compatibility attacks on state-of-the-art cryptography. In *ISOC Network and Distributed System Security Symposium – NDSS 2013*. The Internet Society, February 2013.

[18] Tibor Jager, Sebastian Schinzel, and Juraj Somorovsky. Bleichenbacher's attack strikes again: Breaking PKCS#1 v1.5 in XML encryption. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *ESORICS 2012: 17th European Symposium on Research in Computer Security*, volume 7459 of *Lecture Notes in Computer Science*, pages 752–769. Springer, September 2012.

[19] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447 (Informational), February 2003.

[20] B. Kaliski. PKCS #1: RSA Encryption Version 1.5. RFC 2313 (Informational), March 1998. Obsoleted by RFC 2437.

[21] B. Kaliski and J. Staddon. PKCS #1: RSA Cryptography Specifications Version 2.0. RFC 2437 (Informational), October 1998. Obsoleted by RFC 3447.

[22] Vlastimil Klíma, Ondrej Pokorný, and Tomás Rosa. Attacking RSA-based sessions in SSL/TLS. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 426–440. Springer, September 2003.

[23] Robert Lychev, Samuel Jero, Alexandra Boldyreva, and Cristina Nita-Rotaru. How secure and quick is QUIC? Provable security and performance analyses. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 214–231. IEEE Computer Society, 2015.

[24] M. Maher. ATM Signalling Support for IP over ATM - UNI Signalling 4.0 Update. RFC 2331 (Proposed Standard), April 1998.

[25] James Manger. A chosen ciphertext attack on RSA optimal asymmetric encryption padding (OAEP) as standardized in PKCS #1 v2.0. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 230–238. Springer, August 2001.

[26] Nikos Mavrogiannopoulos, Frederik Vercauteren, Vesselin Velichkov, and Bart Preneel. A cross-protocol attack on the TLS protocol. In Yu et al. [34], pages 62–72.

[27] Christopher Meyer and Jörg Schwenk. SoK: Lessons Learned From SSL/TLS Attacks. In *Proceedings of the 14th International Workshop on Information Security Applications*, WISA 2013, Berlin, Heidelberg, August 2013. Springer-Verlag.

[28] Christopher Meyer, Juraj Somorovsky, Eugen Weiss, Jörg Schwenk, Sebastian Schinzel, and Erik Tews. Revisiting SSL/TLS implementations: New bleichenbacher side channels and attacks. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 733–748, 2014.

[29] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *ACM CCS 09: 16th Conference on Computer and Communications Security*, pages 199–212. ACM Press, November 2009.

[30] Ivan Ristić. *Bulletproof SSL and TLS. Understanding and deploying SSL/TLS and PKI to secure servers and web applications.* Feisty Duck, August 2014.

[31] Jim Roskind. Experimenting with QUIC, 2013. `http://blog.chromium.org/2013/06/experimenting-with-quic.html`.

[32] Jim Roskind. QUIC design document, 2013. `https://docs.google.com/a/chromium.org/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34`.

[33] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. *The Second USENIX Workshop on Electronic Commerce Proceedings*, 1996.

[34] Ting Yu, George Danezis, and Virgil D. Gligor, editors. *ACM CCS 12: 19th Conference on Computer and Communications Security*. ACM Press, October 2012.

[35] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-VM side channels and their use to extract private keys. In Yu et al. [34], pages 305–316.

[36] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-tenant side-channel attacks in PaaS clouds. In Ahn et al. [3], pages 990–1003.