# A Cross-Protocol Attack on the TLS Protocol

Nikos Mavrogiannopoulos
KU Leuven
ESAT/SCD/COSIC – IBBT
Leuven, Belgium
nikos@esat.kuleuven.be

Frederik Vercauteren
KU Leuven
ESAT/SCD/COSIC – IBBT
Leuven, Belgium
fvercaut@esat.kuleuven.be

Vesselin Velichkov
University of Luxembourg
Luxembourg
vesselin.velichkov@uni.lu

Bart Preneel
KU Leuven
ESAT/SCD/COSIC – IBBT
Leuven, Belgium
preneel@esat.kuleuven.be

## ABSTRACT

This paper describes a cross-protocol attack on all versions of TLS; it can be seen as an extension of the Wagner and Schneier attack on SSL 3.0. The attack presents valid explicit elliptic curve Diffie-Hellman parameters signed by a server to a client that incorrectly interprets these parameters as valid plain Diffie-Hellman parameters. Our attack enables an adversary to successfully impersonate a server to a *random* client after obtaining $2^{40}$ signed elliptic curve keys from the original server. While attacking a specific client is improbable due to the high number of signed keys required during the lifetime of one TLS handshake, it is not completely unrealistic for a setting where the server has high computational power and the attacker contents itself with recovering one out of many session keys. We remark that popular open-source *server* implementations are not susceptible to this attack, since they typically do not support the explicit curve option. Finally we propose a fix that renders the protocol immune to this family of cross-protocol attacks.

## Categories and Subject Descriptors

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

## Keywords

SSL, TLS, man-in-the-middle, cross-protocol attack, server impersonation attack

## 1. INTRODUCTION

### *The TLS protocol.*

TLS is one of the major secure communications protocols on the Internet, used by a variety of applications such as web browsers, electronic mail, voice over-IP and more. The protocol derives from Netscape's SSL 3.0 [14], and is the joint attempt, under the umbrella of IETF, to create a secure protocol for e-commerce. The first version of the protocol, TLS 1.0, fixed the known issues [4] in SSL 3.0 and introduced HMAC [17]. TLS 1.1 followed to address known attacks in CBC encryption mode [2, 26] and RSA [16]. Today the latest version is TLS 1.2 [11] but since none of the known weaknesses are classified as major, older versions of the protocol including SSL 3.0 are still in use.

The TLS protocol is an agile protocol that allows peers to negotiate their highest supported protocol version, as well as the combination of ciphers used in a session. That combination is called a ciphersuite; it typically determines the symmetric encryption cipher with its operational mode, the key exchange method and the message authentication algorithm. The various versions of the protocol added new ciphersuites, deprecated old ones, or kept the same set of the previous version.

Not all available ciphersuites in TLS are equally strong: the most prominent example were the ones marked as export ciphersuites. When strong cryptography was not allowed to be exported from USA, major TLS-enabled web browsers that originated in USA included ciphersuites known to be weak. These supported the RSA-EXPORT key exchange method, which used 512-bit RSA keys, and was combined with 40-bit or 56-bit symmetric encryption. Fortunately they were deprecated since TLS 1.1 in 2006 [10].

As a result of the various ciphersuites available in the protocol, a typical implementation includes several algorithms offering similar functionality. For example an implementation may support the Diffie-Hellman (DH) key exchange algorithm, as well as elliptic curve Diffie-Hellman (ECDH). This fact is exploited in our attack by taking advantage of interactions between the different ciphersuites. In particular we exploit the possibility that a client interprets signed ECDH key exchange parameters as plain DH parameters.

### *The Wagner and Schneier attack.*

Wagner and Schneier describe in [27] a server impersonation attack on the SSL 3.0 [14] protocol. Although this attack turned out to be impossible in practice due to an incorrect interpretation of the protocol, the underlying idea is still worth recalling. The attack transforms a server into an or-


*CCS'12,* October 16–18, 2012, Raleigh, North Carolina, USA.

acle that signs messages submitted by the adversary. In particular the server is used by the adversary to sign DH parameters, which are presented to the client as RSA parameters. This allows the recovery of the client's secret by the adversary and eventually to the establishment of a secure session between the adversary and the client. In that session the client is convinced that the adversary is the server he intended to connect to. We will use the term cross-protocol attack to describe this attack, as well as the family of attacks that rely on interactions between distinct key exchange methods. The term multi-protocol attack is also used in the literature to describe this family of attacks [7].

Even if the Wagner and Schneier attack turned out to be impossible to implement, it demonstrates that the TLS protocol violates the following principle set forth by Anderson and Needham in [1].

> **Principle 3**: Be careful when signing or decrypting data that you never let yourself be used as an oracle by your opponent.

This weakness was ignored, possibly, because the only published attack could not be implemented, and required the client to request the deliberately weakened RSA-EXPORT key exchange method.

### *Our attack.*

As the protocol evolved and various other key exchange methods such as SRP, PSK or ECDH [3, 13, 25] were added, the fact that the server can be used as an oracle becomes relevant again. In this paper we re-examine the Wagner and Schneier attack in the context of the latest TLS protocol version [11] and describe a new cross-protocol attack. The attack uses the interactions between DH and ECDH key exchanges. It is based both on the ability to transform a TLS server into an oracle that provides signed parameters, and on TLS implementations blindly trusting those signed parameters.

Our contributions in this paper are as follows.

- To our knowledge, our attack is the first server impersonation attack on the TLS protocol with complexity much lower than a cryptanalytic attack on the cryptographic primitives used;
- Our attack highlights a much larger family of cross-protocol attacks that the TLS protocol is vulnerable to, which was previously ignored;
- We show that although basic checks on DH protocol parameters help to mitigate simple attacks, they are not sufficient to completely protect the protocol.

### *The adversary.*

The adversary in both attacks is a Dolev-Yao adversary [12], that has full control over the network communications.

### *Paper organization.*

In Section 2 we present the Wagner and Schneier attack on SSL 3.0 and the incorrect assumption that renders the attack impossible. Then in Section 3 we present our attack on the protocol, and in Section 4 we discuss the impact of the attack on several implementations. Section 5 provides a simulation of the attack in a real world scenario, and in Section 6 we propose a fix that makes TLS immune to this family of attacks. Finally Section 7 concludes the paper.

### *Terminology.*

This document assumes familiarity with the TLS protocol [11] and adopts its terminology. Furthermore, when we refer to Diffie-Hellman key exchange we denote with $g$ the generator of the multiplicative group modulo $p$, and with $Y_s$ and $Y_c$ the public values of the server and the client. The explicit elliptic curves supported by TLS are given by a Weierstrass equation of the form

$$y^2 = x^3 + ax + b \mod q\,.$$

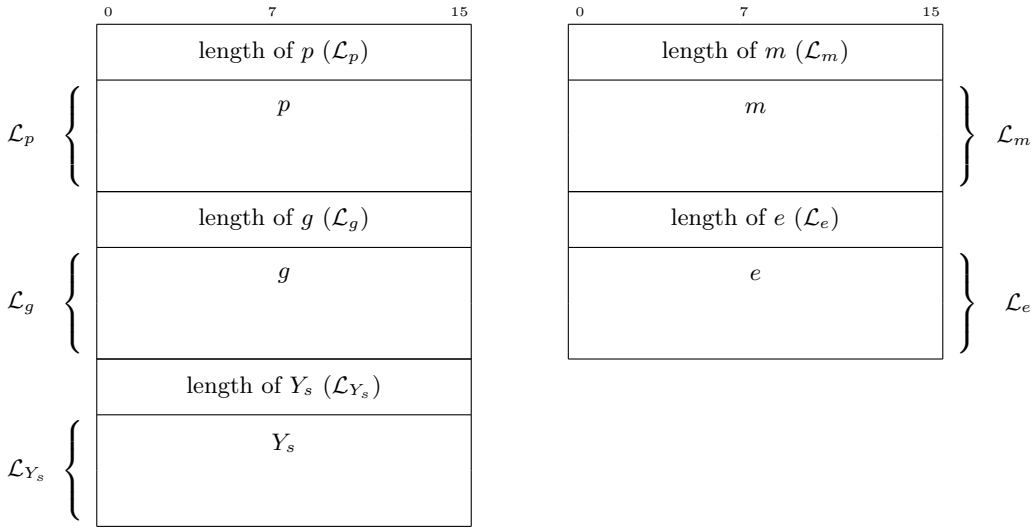
Note that we use $q$ to denote the ECDH prime to distinguish it from the plain DH prime $p$. The coefficients $a$ and $b$ are the curve parameters represented as integers modulo $q$. The protocol works in a group generated by a base point $P$ (simply called base in the remainder of the paper). The cofactor is defined as the order of the curve (i.e. the number of points on the curve) divided by the order of the base point. A public ECDH share is of the form $Q = [k]P = (X, Y)$, with $k$ the private key, $Q$ the elliptic curve point obtained by scalar multiplication of $P$ by $k$ and $X$ (resp. $Y$) the $x$ (resp. $y$) coordinate of $Q$.

## 2. THE WAGNER AND SCHNEIER ATTACK

Wagner and Schneier in [27] describe a cross-protocol attack (the authors refer to it as "key exchange algorithm rollback attack") based on the observation that the digital signature in a DH key exchange does not cover any identifier of the negotiated ciphersuite. According to the SSL 3.0 protocol [14] when a DH key exchange has been negotiated, the group parameters and key exchange data are sent by the server in the 'ServerKeyExchange' message as shown in Fig. 1a. The signature on that message is calculated on the algorithm parameters, and the nonces exchanged by both peers. The crucial observation is that the negotiated key exchange method is not part of this signature.

This omission allows an adversary to re-use a signed 'ServerKeyExchange' packet in another session, with another key exchange method, by initiating a parallel connection to the server. The attack deceives a client who advertises a 'TLS_RSA_EXPORT' ciphersuite and expects temporary RSA parameters in the 'ServerKeyExchange' message, into receiving DH parameters from a 'TLS_DHE_RSA' ciphersuite. Note that, the RSA-EXPORT key exchange requires the server to generate a temporary 512-bit long RSA key pair and include it in the 'ServerKeyExchange' message. In both DH and RSA-EXPORT the parameters are signed using the RSA algorithm.

The attack assumes that the client reads and verifies the signature, and then reads the RSA parameters (see Fig. 1b) one by one, yielding the following scenario. The client verifies the signature, reads the RSA modulus $m$, which corresponds to the prime of the DH group $p$, and then reads the RSA exponent $e$ field which corresponds to the group generator $g$. Therefore, the client encrypts the pre-master secret $k$ as $k^g \mod p$ and includes it in its 'ClientKeyExchange' message. Since $p$ is a prime number and $g$ is known, it is very easy to compute the $g$-th root of $k^g$ to recover $k$, which allows the adversary to impersonate the server. Note that the 'Finished' messages that provide handshake message modification detection using message hashes encrypted and authenticated with the session keys, cannot detect this attack since the adversary recovers the pre-master secret.




**(a) Diffie-Hellman** **(b) RSA-EXPORT**

**Figure 1: The contents of the ServerKeyExchange message in Diffie-Hellman and RSA-EXPORT key exchange methods. Each row represents a 2-byte (16-bit) field, unless the length is explicitly given. All indicated lengths are in bytes.**

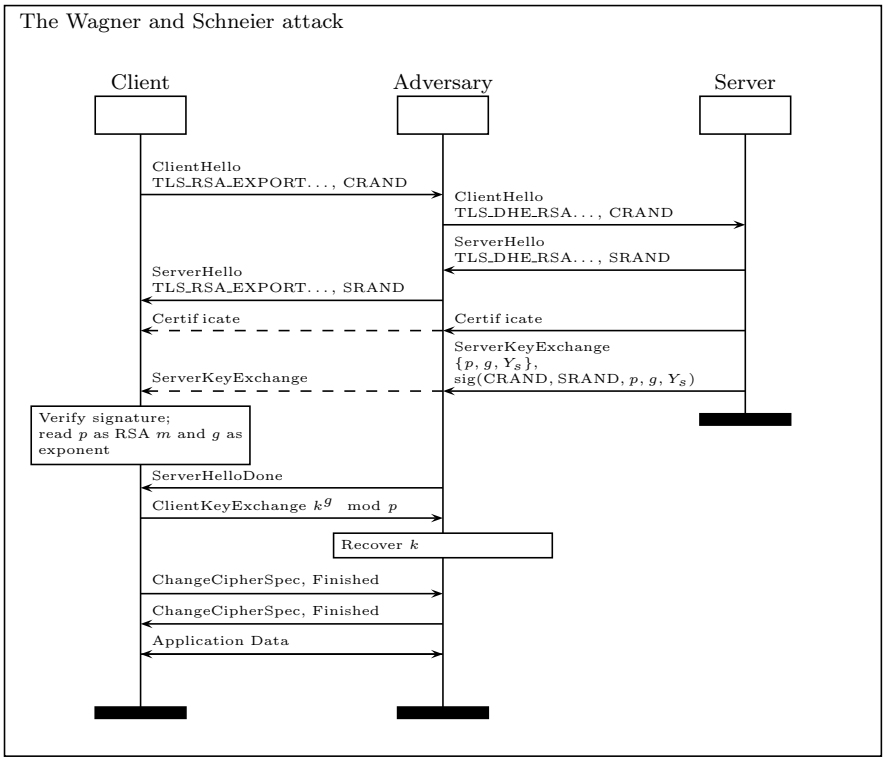



**Figure 2: A message sequence chart describing the Wagner and Schneier attack on SSL 3.0. The dashed lines indicate a forwarded message.**

The authors noticed that the SSLRef 3.0b1 implementation was immune to the attack and attributed the failure to a paranoid sanity check of this particular implementation. However, tests we performed on other implementations which did not include such sanity checks also failed. Careful examination of the TLS packet parsing reveals that the failure of the attack is due to the serialized way TLS packets need to be parsed. The variable length vectors [14] used in the structure definition in Fig. 1 require an implementation to read the vector length before reading data, hence an implementation can only start parsing the packet serially, i.e., from start to end without being able to read a field before reading the previous one. In the RSA case, a client would need to read the modulus length, then the modulus, and the same for the exponent and signature fields. If the DH 'ServerKeyExchange' packet, which contains one additional field, is substituted, that field will be read instead of the signature and verification fails.

Even though the Wagner and Schneier attack fails, it demonstrates the idea of a cross-protocol attack utilizing two of the SSL 3.0 key exchange methods, the DH key exchange and the RSA-EXPORT key exchange.

# 3. A NEW CROSS-PROTOCOL ATTACK

Since version 1.0 [9] the TLS protocol has been augmented with various other key exchange methods such as SRP, PSK or ECDH [3, 13, 25]. In this section we present a server impersonation attack on clients that support the DH key exchange and wish to connect to a server that supports, among others, the ECDH method.

In order to support the ECDH key exchange method, the 'ServerKeyExchange' message was augmented in [3] to allow for several elliptic curve ciphersuites supporting multiple sub-options. The sub-option relevant to this paper is the representation of the elliptic curve used. It allows for explicit prime curves, explicit curves of characteristic 2, or named curves. Depending on the negotiated ciphersuite the structure containing the selected curve parameters is signed by the server using an RSA or ECDSA key. The attack we present requires the server to support the explicit prime curve option, and the client to support the plain DH method. Because the only common signature algorithm in the ECDH and DH key exchanges is RSA, the server is also required to possess an RSA signing key.

## 3.1 Summary of the attack

In the explicit prime curve option, the server includes in its signed 'ServerKeyExchange' message the parameters of its elliptic curve and an ephemeral public key to be used for this session (see Fig. 3a). The randomness of the public key contributes to the feasibility of achieving a cross-protocol attack.

In the attack the adversary, after receiving the client's Hello message, initiates multiple connections to the server, until an ECDH 'ServerKeyExchange' is presented that satisfies two properties. The first is that the message can be interpreted as a valid DH 'ServerKeyExchange' message, and secondly, the adversary can recover the exchanged DH key. After a suitable message is received, the adversary forwards it to the client, who verifies the (valid) signature and proceeds with the handshake. Assuming the adversary can recover the exchanged DH key, the handshake with the client completes and thus the server impersonation is successful. The attack is sketched in Fig. 4.

We first estimate the probability with which a valid ECDH key exchange message can be interpreted as a valid DH key exchange message. Then we investigate how the adversary can recover the session key, either by explicitly computing a discrete logarithm or forcing the session key to take a value in a limited set. Finally, we compute the number of server connections required by the simplest version of our attack.

## 3.2 Probability of valid key exchange message

*Length requirements on key exchange parameters.*

The attack success depends on whether the signed ECDH parameters can be interpreted as DH parameters. In Fig. 3 we contrast the contents of the 'ServerKeyExchange' packets in both cases. The ECDH parameters consist of the constant curve parameters followed by the randomly generated ECDH public key. For our attack to succeed, we require that the $p$ field in the DH parameters extends past the constant curve parameters[1], which results in $p$ having its least significant bytes and $g$ and $Y_s$ fully positioned in the space of the ephemeral elliptic curve public key and therefore have random contents. This means that multiple queries to such server would provide 'ServerKeyExchange' messages with ECDH parameters that if interpreted as DH parameters will have variable lengths for $g$ and $Y_s$ and there is a non-zero probability for these lengths to have valid values (i.e. add up to the remaining message length).

Before calculating this probability, we need to clarify when these length fields are positioned in the ephemeral public key space. As already mentioned, this depends on the length of $p$, which is interpreted based on the contents of the curve_type and elliptic curve prime length ($\mathcal{L}_q$) as shown in Fig. 3a. Since in the explicit curves TLS option the curve_type byte contains the identifier 1, the length in bytes of $p$ ($\mathcal{L}_p$) would be interpreted as:

$$\mathcal{L}_p \;=\; 1 \,||\, \mathcal{L}_q \;=\; 256 + \mathcal{L}_q\,, \tag{1}$$

where $||$ denotes concatenation. For the message to be parsed correctly and the $g$ and $Y_s$ lengths to be placed accordingly, $\mathcal{L}_p$ must be larger than the length of all the fixed parameters and less than the total length of the ECDH parameters (minus the minimum length of the $\mathcal{L}_g, g, \mathcal{L}_{Y_s}, Y_s$ fields being 6 bytes). The fixed parameters are the prime $q$ and the fields marked as $C$ in the ECDH message as shown in Fig. 3a. Then:

$$C = \mathcal{L}_a + \mathcal{L}_b + \mathcal{L}_{\text{base}} + \mathcal{L}_{\text{order}} + \mathcal{L}_{\text{cofactor}} + 7 \tag{2}$$

and we require:

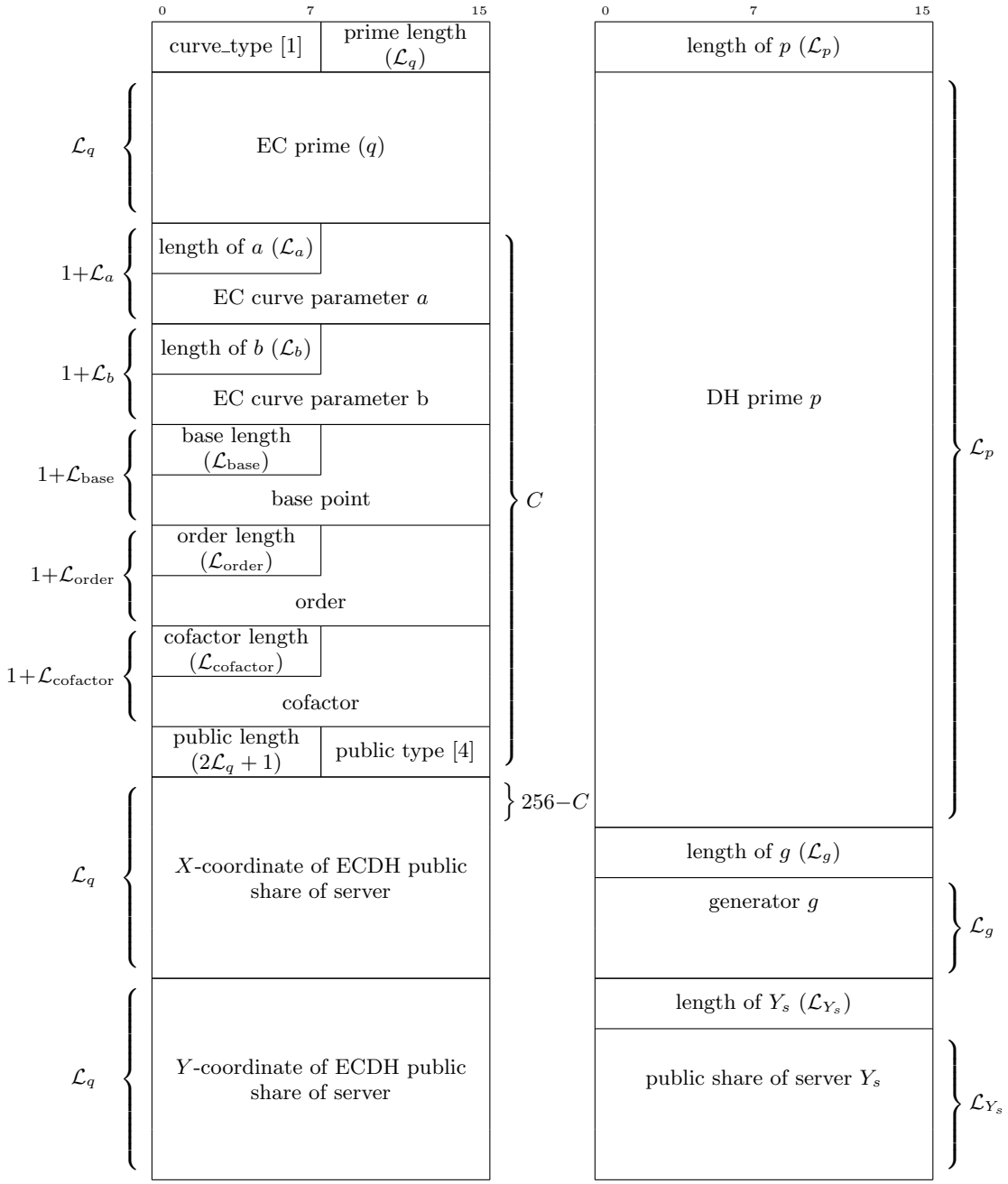
$$\mathcal{L}_q + C \leq \mathcal{L}_p \leq C + 3\mathcal{L}_q - 6$$

Note that in the above equation, $3\mathcal{L}_q$ represents the sum of the lengths of the prime $q$, and the $X$ and $Y$-coordinates of the ephemeral elliptic curve public key. Combined with Eqn. (1) we conclude that $C$ should satisfy:

$$262 - 2\mathcal{L}_q \leq C \leq 256\,. \tag{3}$$

The constraints for $C$ are not unrealistic, and if we consider randomly generated curves (i.e. with random $a$ and $b$) the

[1]If this requirement is not satisfied an attack may be possible on servers with constant curve parameters of certain form. We do not consider this attack in this paper.




**(a) Elliptic Curve Diffie-Hellman** **(b) Diffie-Hellman**

**Figure 3: Contrasting the 'ServerKeyExchange' message contents with Diffie-Hellman and explicit elliptic curve Diffie-Hellman parameters, side-by-side. Each row represents a 2-byte (16-bit) field, unless the length is explicitly given. All the indicated lengths are in bytes and the numbers in brackets denote a constant value field.**

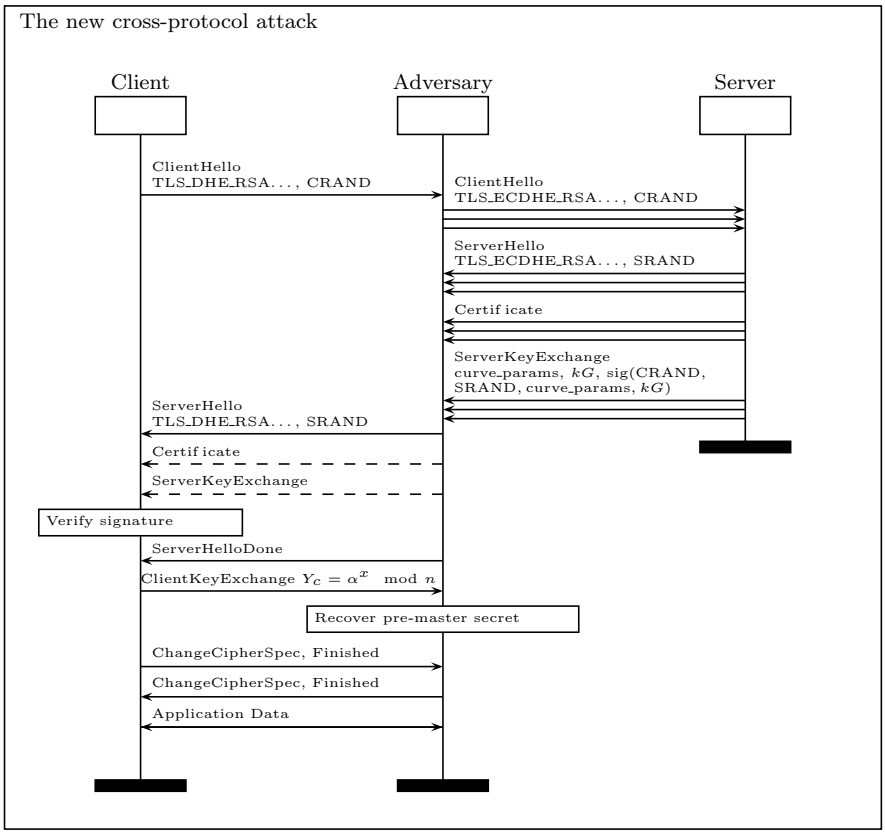



**Figure 4: A message sequence chart describing our attack. The dashed lines indicate a forwarded message, and the consequent lines indicate multiple trials.**

**Table 1: The sizes (in bytes) of the parameters in various named curves. The curve marked with gray fulfills the requirements for the attack: $262 - 2\mathcal{L}_q \leq C \leq 256$.**

| Curve name | $\mathcal{L}_q$ | $\mathcal{L}_a$ | $\mathcal{L}_b$ | $\mathcal{L}_{\text{base}}$ | $\mathcal{L}_{\text{order}}$ | $\mathcal{L}_{\text{cofactor}}$ | $C$ | $262 - 2\mathcal{L}_q$ |
|---|---|---|---|---|---|---|---|---|
| secp192k1 | 24 | 1 | 1 | $2\mathcal{L}_q + 1$ | $\mathcal{L}_q$ | 1 | 83 | 214 |
| secp192r1 | 24 | $\mathcal{L}_q$ | $\mathcal{L}_q$ | $2\mathcal{L}_q + 1$ | $\mathcal{L}_q$ | 1 | 129 | 214 |
| secp224k1 | 28 | 1 | 1 | $2\mathcal{L}_q + 1$ | $\mathcal{L}_q$ | 1 | 95 | 206 |
| secp224r1 | 28 | $\mathcal{L}_q$ | $\mathcal{L}_q$ | $2\mathcal{L}_q + 1$ | $\mathcal{L}_q$ | 1 | 149 | 206 |
| secp256k1 | 32 | 1 | 1 | $2\mathcal{L}_q + 1$ | $\mathcal{L}_q$ | 1 | 107 | 198 |
| secp256r1 | 32 | $\mathcal{L}_q$ | $\mathcal{L}_q$ | $2\mathcal{L}_q + 1$ | $\mathcal{L}_q$ | 1 | 169 | 198 |
| secp384r1 | 48 | $\mathcal{L}_q$ | $\mathcal{L}_q$ | $2\mathcal{L}_q + 1$ | $\mathcal{L}_q$ | 1 | 249 | 166 |
| secp521r1 | 66 | $\mathcal{L}_q$ | $\mathcal{L}_q$ | $2\mathcal{L}_q + 1$ | $\mathcal{L}_q$ | 1 | 339 | 130 |

attack will work for any elliptic curve over a prime finite field of characteristic roughly between 300 and 400 bits. For instance, if we test the prime curves listed in [6] we see that they are fulfilled for the secp384r1 curve (although this is a named curve), as shown in Table 1.

### *Probability estimate.*

If a server uses explicit elliptic curve parameters in the appropriate range, the attack is straightforward, even though it requires quite some effort from the adversary and the server. The adversary intercepts a client connection; upon receipt of the 'ClientHello', initiates multiple connections to the server. His goal is to obtain signed ECDH parameters that contain valid lengths for $g$ and $Y_s$.
From Fig. 3a we see that the maximum valid length of $g$ is given by

$$2\mathcal{L}_q - (256 - C) - 2 - 2 - 1\,,$$

where the consecutive terms in the above sum correspond to: the size of $X$ and $Y$-coordinate, the part taken up by $p$, the length field of $g$, the length field of $Y_s$ and the minimal length of $Y_s$. If we define this upper bound as $L := 2\mathcal{L}_q - 261 + C$, then we conclude that the valid lengths of $g$ satisfy

$$0 < \mathcal{L}_g \leq L\,. \tag{4}$$

For each valid $\mathcal{L}_g$ there is precisely one possible length of $Y_s$, namely

$$\mathcal{L}_{Y_s} = L + 1 - \mathcal{L}_g\,. \tag{5}$$

Since the lengths of $g$ and $Y_s$ are 16-bit values and assuming a uniform distribution of the bytes on the elliptic curve public key, the probability of the ECDH message being parsed successfully as DH parameters is:

$$P(\text{valid message}) = \frac{L}{2^{16}} \cdot \frac{1}{2^{16}} = \frac{L}{2^{32}}\,. \tag{6}$$

Since the lengths of $g$ and $Y_s$ are assumed random, the probabilities that these attain any given fixed value are independent. The uniform distribution assumption for the values read as the $g$ and $Y_s$ lengths may appear questionable. Even though a random $X$-value has probability close to $1/2$ of being a valid $X$-coordinate and its uniformity assumption is plausible, this is not the case with the $Y$-coordinate. To confirm this assumption we calculated using [21] the estimated PDF, shown in Fig. 5, for the two lengths (of $g$ and $Y_s$) on the secp384r1 curve. The plots strengthen the assumption of uniformity. Furthermore, by simulating the attack on the secp384r1 curve, we calculated the empirical probability for a valid message and compared it with the theoretical probability computed by Eqn. (6), in Table 2. The theoretical values do not significantly deviate from the empirical ones.

**Table 2: The theoretical and empirical probabilities of finding a valid Diffie-Hellman key exchange message using the explicit parameters of the secp384r1 curve.**

| Probability | Theoretical | Empirical |
|---|---|---|
| valid message | $1.9 \cdot 10^{-8}$ | $2.0 \cdot 10^{-8}$ |

## 3.3 Recovering the session key

So far, we have explained how the adversary can use the server as an oracle to receive, with a certain probability,

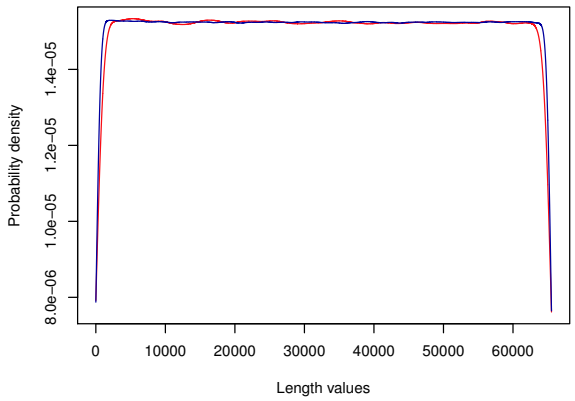



**Figure 5: The estimated PDF for values read as the length of $g$ (blue), and the length $Y_s$ (red) on the secp384r1 curve. The length of $Y_s$ is calculated after a suitable length for $g$ is found.**

a signed message containing a triplet of random numbers, which the client interprets as the DH parameters ($p$, $g$, $Y_s$). To prevent confusion with the role implied by these symbols (e.g. $p$ may not be a prime) we will use the symbols $(n, \alpha, \beta)$ instead.
Upon receipt, the client calculates his DH share as $Y_c \equiv g^x \equiv \alpha^x \mod n$, and the pre-master secret as $Y_s^x \equiv \beta^x \mod n$. Recall from Eqn. (1) that the size of $n$ is larger than 256 bytes or 2048 bits. To compute the session key, the adversary can either recover $x$ from $Y_c$ exploiting the fact that $n$ is not necessarily prime or he can force specific values for the pre-master secret $\beta^x$ by judiciously selecting the triplet $(n, \alpha, \beta)$ he forwards to the client. We now analyze both cases in detail.

### *Computing $x$.*

Recovering $x$ from $\alpha^x \bmod n$ corresponds to computing a discrete logarithm in $\mathbb{Z}_n$. If the factorization of $n = \prod_{i=1}^{s} p_i^{e_i}$ is known or can be computed, this problem can be easily reduced (using the Chinese Remainder Theorem) to a number of discrete logarithm problems in $\mathbb{F}_{p_i}$. Unfortunately, computing the factorization of $n$ using the Number Field Sieve [19] (NFS) has similar complexity to computing discrete logarithms in $\mathbb{F}_p$ with $p$ the same size as $n$. On the other hand, the complexity of the Elliptic Curve factoring method [20] (ECM) depends on the size of the smallest prime factor of $n$, so if the $p_i$ are much smaller than $n$, ECM would outperform NFS easily. To analyze the applicability of ECM, recall that a number $n$ has prime factors $p_i$ all smaller than $n^{1/u}$ with probability asymptotically equal to $u^{-u}$ [8].
Two different attack scenarios now arise, namely online vs. offline. In the online scenario, the attacker has to compute $x$ during the lifetime of the handshake. Since the typical handshake timeout is less than a minute, the attacker can only succeed if $n$ is really smooth. Since $n$ has more than 2048 bits, we would require $u > 20$ for the prime factors to be small enough for the attack to succeed within the timeout.

Hence, the probability that $n$ satisfies these requirements is $< 2^{-80}$.
In the offline scenario, the attacker has a much higher probability of succeeding. Indeed, now $n$ can have one large prime factor (e.g. around 530 bits, the current DLP record [15]) and all other prime factors much smaller (e.g. around 240 bits, the current ECM record [5]). Since the size of $n$ is never larger than 2500 bits, the probability that the attacker succeeds given a valid message is larger than $8^{-8} = 2^{-24}$. This probability then needs to be multiplied with Eqn. (6) which is around $2^{-25}$. The overall success probability therefore is around $2^{-49}$.
The fact that the key can be recovered offline does not pose any threat to the TLS protocol since in the normal execution of the protocol, the application data are only transmitted after the verification of 'Finished' messages by both peers. However, extensions to the protocol such as the "False start" [18] that try to reduce protocol round-trips by sending the client's application data before the peer's finished message is verified, are at risk. This attack may be used to obtain the encrypted client's data and decrypt it by calculating the shared key offline.

### *Computing pre-master secret.*

Recall that the ephemeral key $Y_s$ is given by part of the X and/or Y coordinate of the ephemeral public key generated by the server. It is thus possible and in fact not unlikely (see the next section), that $Y_s = 0, \pm 1$. As observed in [22], if the client accepts any of these values, the pre-master secret is very easy to compute since it will be equal to $0, \pm 1$.
Such values for $Y_s$ can be generalized slightly when there are other roots of unity of small order or roots of nullity if $n$ is composite, since then it becomes likely that the pre-master secret again equals 1 or 0. The main obstacle for this generalisation is the fact that the available length for $Y_s$ is rather small compared to the total length of $n$. We require roots of unity and/or nullity with compact representation modulo $n$. For roots of nullity it is easy to see that this is very unlikely: indeed, let $n = \prod_{i=1}^{s} p_i^{e_i}$, then the root of nullity with most compact representation modulo $n$ is given by $r = \prod_{i=1}^{s} p_i$. This shows that a small root of nullity is extremely unlikely. For roots of unity of order $k > 2$, the condition is that $n \mid \Phi_k(r)$, with $\Phi_k(x)$, the $k$-th cyclotomic polynomial, so for random $n$ this is again unlikely. Hence, we will mainly focus on the case $Y_s = 1$.

## 3.4 Attack success probability

In this section, we provide a lower bound on the adversary's probability of success for the simplest attack where the adversary expects to find the value 1 in $Y_s$.
To compute the probability that $Y_s = 1$ for a valid message, we again refer to Fig. 3. For every fixed valid length $\mathcal{L}_g = k$ with $0 < k \leq L$ we can compute the probability that the message is valid and $Y_s = 1$. Recall that for $\mathcal{L}_g = k$, we have $\mathcal{L}_{Y_s} = L + 1 - k$. If we set $V_k$ to be the event that $\mathcal{L}_g = k$ and $\mathcal{L}_{Y_s} = L + 1 - k$, then the probability that $V_k$ and $Y_s = 1$ (all bytes of $Y_s$ equal to zero, except the least significant byte equal to 1) is therefore given by

$$P(V_k \wedge Y_s = 1) = P(V_k)P(Y_s = 1 \mid V_k) = \frac{1}{2^{32}} \cdot \frac{1}{2^{8(L+1-k)}}.$$

Summing over all $0 < k \leq L$ we thus obtain

$$P(\text{valid message} \wedge Y_s = 1) = \frac{1}{2^{32}} \sum_{i=1}^{L} \frac{1}{2^{8i}} \approx 2^{-40}. \quad (7)$$

This shows that an adversary needs roughly $2^{40}$ signed elliptic curve public keys from the server to obtain a message that satisfies the attack requirements. For the attack to proceed, we note that this message needs to be received during the lifetime of the TLS handshake with a client (this could be a random client).

# 4. ATTACK ASSUMPTIONS

The described attack relies on the following two assumptions.

1. The client software supports one of the 'TLS_DHE_RSA' ciphersuites and a DH public key ($Y_s$) with value 1 is accepted;
2. The server software supports one of the 'TLS_ECDHE_RSA' ciphersuites, with the 'arbitrary_explicit_prime_curve' [3] option, has selected a curve of size between 300 and 400 bits and uses RSA as the signing algorithm.

### *First assumption.*

The first assumption turns out to be true for several implementations. We tested TLS implementations that support the 'TLS_DHE_RSA' ciphersuites, and concluded that not all of them include sanity checks on the received DH parameters. We tested the invalid values $0, \pm 1$ and summarized the obtained results in Table 3.

**Table 3: The behavior of implementations when receiving various values as a Diffie-Hellman public key ($Y_s$).**

| | $Y_s = 1$ | $Y_s = 0$ | $Y_s = -1$ |
|---|---|---|---|
| NSS 3.12.6 | Accept | Accept | Accept |
| OpenSSL 1.0.1 | Reject | Reject | Reject |
| GnuTLS 3.0.18 | Accept | Accept | Accept |

We can see that there are implementations that blindly trust the signed parameters received by the server and do not check for invalid parameters[2]. Note that the value of 0 in NSS was rejected if it had minimal encoding but accepted otherwise.
It can be argued that these are individual implementation bugs because the TLS protocol [11] includes recommendations for checking DH parameters. However the simple checks for public values of $\pm 1$ and 0 are not explicitly mentioned and the implementer is directed to other documents instead. Although there is merit in testing for obvious invalid parameters, the protocol should not rely on these tests. Such tests will rule out corner cases like the one our attack is using, but no matter how thorough they cannot detect values like roots of unity or nullity that can be used to mount a variant of our attack, nor ensure the proper generation of the parameters. It can be argued that the former values could be ruled-out by forcing the client to perform a primality check on the group modulus and to calculate the order of the generator. However, even if that was an acceptable time

[2]Note that the authors of these implementations have been contacted and the described vulnerabilities have been fixed.

to security trade-off, the latter cannot be performed with the information provided in a TLS handshake. It is crucial that the trust in the peer's signature needs to extend to a trust in the parameters, and that can only be achieved with a protocol fix.

*Second assumption.*
Although almost all servers on the Internet have an RSA public key, at the moment this paper was written, public data available about Internet servers such as the 'Internet SSL Survey' [24] did not include the required information to assess whether servers supporting arbitrary elliptic curves exist. Our belief is that because popular open source TLS protocol implementations such as NSS, OpenSSL and GnuTLS currently do not support arbitrary elliptic curves, the majority of Internet servers are unaffected by our attack.

# 5. FEASIBILITY OF THE ATTACK

Given the large number ($2^{40}$) of signed messages the adversary requires from the server while the client's handshake is on hold, we need to evaluate the feasibility of the attack in a real world scenario. For that we performed a simulation of the part of the attack involving the server and the adversary. However, since open-source implementations of TLS do not support explicit elliptic curves, in the simulation we use the named curve TLS option, which limits our choice of curves. In our simulation, the server is a web-server that supports TLS with the secp384r1 curve and holds a 1024-bit RSA key. The adversary performs multiple partial handshakes with the server, that are terminated once the signed 'ServerKeyExchange' message is received. The simulation was performed using two 24-CPU Intel Xeon (X5670) systems interconnected using Gigabit Ethernet. One was used as an HTTPS server using the nxweb[3] software with GnuTLS 3.0.20, and the other was used to initiate the adversary's connections to the server. The software used to simulate the adversary is a modified version of the httpress[4] tool, that measures the number of requests per second of the partial TLS handshake used in the attack. Both nxweb and and httpress operate in a multi-threaded manner and were assigned all the available CPUs in each system.
We now consider two variants of the attack. The first targets a specific client, following closely our attack as in Fig. 4, and the second is an attack that targets any client.

## 5.1 Attacking a specific client

In this attack the adversary's goal is to impersonate the server to a specific client of his choice. The adversary would then be required to perform $2^{40}$ connections to the server before the client times out. In our test setup the adversary was able to perform 3770 connections per second on average, resulting in an expected time of approximately 9 years to attempt all the $2^{40}$ connections. This is quite a long time given the TLS handshake timeout values in browsers (see Table 4), and human patience. However, typical high-load Internet servers operate in clusters and may support hardware acceleration of RSA signing operations. On such a combination that improves performance by a factor of 1000 the estimated time is 3 days. That is still a long time for a single interactive session, indicating that the described attack may not be practical today in attacking a specific targeted client. The attack simulation results are summarized in Table 5.

[3] `https://bitbucket.org/yarosla/nxweb/wiki/Home`
[4] `https://bitbucket.org/yarosla/httpress/wiki/Home`

**Table 4: TLS handshake timeout values in various browsers.**

| Browser | Handshake timeout |
|---|---|
| Chrome 20 | 20 secs |
| Firefox 10 | 30 secs |
| Internet Explorer 8 | 40 secs |
| Opera 12 | 40 secs |

**Table 5: The resources required by the web server during the attack simulation.**

| Web server | |
|---|---|
| Transmitted data | 4.7 MB/sec |
| Received data | 1.8 MB/sec |
| Requests handled | 3770 req/sec |

## 5.2 Attacking a random client

On the other hand, there are cases where the adversary is not particularly interested in a specific client and impersonating the server to a random client is sufficient. In that scenario, the adversary hijacks every (distinct) client connection attempt and uses the interval before the connection times out to initiate multiple connections to the server. On every failed attempt he gives up on that client who sees a connection timeout in his browser. Using the numbers from the simulation, and assuming a 40 second connection timeout value, the adversary would expect to impersonate the server after $2^{23}$ distinct client connection attempts. For high-load servers this is a rather low number. For example, Google currently has 700 million unique visitors per day, so it would only take on average 17 minutes for the attack to succeed (if the increase in load can be absorbed).

# 6. A POSSIBLE FIX

Currently the signature on the 'ServerKeyExchange' message covering the key exchange parameters, ensures only the freshness of the message, but not whether it was intended for this particular key exchange. A simple fix may be to include the negotiated ciphersuite into the signature. However, the TLS protocol is complex and negotiates extensions that modify the key exchange in several ways. For example, in this particular attack we used the explicit curves option of TLS, which is negotiated using an extension. Such extensions should also be covered by the signature, to prevent any attack that uses the sub-options allowed by a key exchange algorithm.
For that we propose to modify the signature of the 'ServerKeyExchange' to include, in addition to explicit identifiers of the algorithms, all the previously exchanged messages. Our proposed signature for a 'ServerKeyExchange' message is shown in Fig. 6. It includes explicit indicators of the entity (server), the key exchange algorithm used, the handshake messages exchanged, and the parameters of the key exchange. This modification may be negotiated either with an upgraded TLS version number, or by defining a new TLS

extension similarly to the approach in [23], allowing backwards compatibility.

```
enum { server (0), client (1) } ConnectionEnd;

enum { dhe_dss (0), dhe_rsa (1),
       ec_diffie_hellman (2)
     } KeyExchangeAlgorithm;

struct {
  select (KeyExchangeAlgorithm) {
    case dhe_dss:
    case dhe_rsa:
      ServerDHParams params;
    case ec_diffie_hellman:
      ServerECDHParams params;
  }
} Parameters;

struct {
  Parameters params;
  digitally-signed struct {
    ConnectionEnd entity;
    opaque handshake_messages<1..2^24-1>;
    KeyExchangeAlgorithm kx_algorithm;
    Parameters params;
  }
} ServerKeyExchange;
```

**Figure 6: The proposed format for the ServerKeyExchange message signature. Note that we follow the TLS protocol message description. In particular, the type opaque is used to indicate bytes containing uninterpreted data and arrays of variable length, specified with the <floor..ceiling> notation, are preceded by a number of bytes containing the length of the array.**

A drawback of this change is that it requires caching of the previously exchanged messages until the 'ServerKeyExchange' message is sent. This however, should be an insignificant cost for today's servers.

# 7. CONCLUSIONS

In this paper we presented a new attack on the TLS protocol that exploits the fact that a client can interpret signed explicit elliptic curve Diffie-Hellman (DH) key exchange parameters as valid plain DH parameters. It is a cross-protocol attack similar in nature to the attack introduced by Wagner and Schneier. The attack enables impersonation of the server and is much more efficient than breaking any of the involved cryptographic primitives.

Nonetheless, the presented attack depends on the server supporting the explicit elliptic curves option. This option is not supported in the tested open-source implementations making them resistant to this attack. This fact suggests that for now the implementation of the explicit elliptic curves protocol option should be avoided unless a counter-measure like our proposed fix is in place.

A limiting factor of the attack in servers that support the explicit elliptic curve option is that it requires the initiation of $2^{40}$ sessions within the timeframe of the client session. We show that this may be prohibitive for attacking a specific client, but if the target is attacking any random client, the attack should be feasible on a cluster of servers.

Moreover, we show that extensions of the TLS protocol such as the "False start" [18] that reduces protocol round-trips by sending the encrypted client's application data before the full handshake is complete, are at risk. The attack may be used by an adversary to recover the encryption key offline and access the encrypted data.

The described attack can be countered by verifying the server's DH public key for known invalid values. The TLS protocol, however, should not over-rely on these tests. There could be other values that can be used to mount variants of our attack but cannot be easily detected. We believe that the trust in the peer's signature needs to extend to a trust in the signed parameters.

It is also worth noting that our attack may not be the only possible cross-protocol attack on the TLS protocol. Due to the fact that the protocol in its current form allows a server to be used as an oracle by an adversary, other attacks that explore different algorithm interactions may also be possible. For that we proposed a fix to the protocol that defends against this family of attacks.

# 8. ACKNOWLEDGMENTS

The authors would like to thank David Wagner, Kenny Paterson, Andreas Pashalidis, Yngve Nysaeter Pettersen, Eric Rescorla, Adam Langley, Bodo Moeller, Marsh Ray, and the anonymous referees for their comments which improved this manuscript. Moreover we would like to thank Koen Simoens and Elmar Tischhauser who contributed in the formulation of this attack. This work was supported in part by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT Vlaanderen) SBO project, the Research Council KU Leuven: GOA TENSE (GOA/11/007), by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy).

# 9. REFERENCES

[1] R. J. Anderson and R. M. Needham. Robustness principles for public key protocols. In *CRYPTO 1995*, volume 963 of *Lecture Notes in Computer Science*, pages 236–247. Springer, 1995.

[2] G. V. Bard. A challenging but feasible blockwise-adaptive chosen-plaintext attack on SSL. In *SECRYPT 2006*, pages 99–109. INSTICC Press, 2006.

[3] 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), 2006.

[4] D. Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. In *CRYPTO 1998*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1998.

[5] J. Bos, T. Kleinjung, A. Lenstra, and P. Montgomery. This is a tasty factor. Email on NMBRTHRY-list. 8 Mar 2010.

[6] Certicom Research. SEC 2: Recommended elliptic curve domain parameters, September 2000.

[7] C. J. F. Cremers. Feasibility of multi-protocol attacks. In *ARES 2006*, pages 287–294. IEEE Computer Society, 2006.

[8] K. Dickman. On the frequency of numbers containing prime factors of a certain relative magnitude. *Arkiv för Matematik, Astronomi och Fysik*, 22:1–14, 1930.

[9] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), 1999.

[10] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), 2006.

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

[12] D. Dolev and A. C.-C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[13] P. Eronen and H. Tschofenig. Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279 (Proposed Standard), 2005.

[14] A. Freier, P. Karlton, and P. Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101 (Historic), 2011.

[15] T. Kleinjung. Discrete logarithms in GF(p) — 160 digits. Email on NMBRTHRY-list. 5 Feb 2007.

[16] V. Klíma, O. Pokorný, and T. Rosa. Attacking RSA-Based Sessions in SSL/TLS. In *CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 426–440. Springer, 2003.

[17] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), Feb. 1997.

[18] A. Langley, N. Modadugu, and B. Moeller. Transport Layer Security (TLS) False Start. Internet Draft, 2010.

[19] A. K. Lenstra and H. W. Lenstra, Jr., editors. *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1993.

[20] H. W. Lenstra, Jr. Factoring integers with elliptic curves. *Annals of Mathematics (2)*, 126(3):649–673, 1987.

[21] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. ISBN 3-900051-07-0.

[22] J.-F. Raymond and A. Stiglic. Security issues in the Diffie-Hellman key agreement protocol. *IEEE Transactions on Information Theory*, 22:1–17, 2000.

[23] E. Rescorla, M. Ray, S. Dispensa, and N. Oskov. Transport Layer Security (TLS) Renegotiation Indication Extension. RFC 5746 (Proposed Standard), 2010.

[24] I. Ristić. Internet SSL Survey, 2011. `https://www.ssllabs.com/projects/ssl-survey/`.

[25] D. Taylor, T. Wu, N. Mavrogiannopoulos, and T. Perrin. Using the Secure Remote Password (SRP) Protocol for TLS Authentication. RFC 5054 (Informational), 2007.

[26] S. Vaudenay. Security Flaws Induced by CBC Padding– Applications to SSL, IPSEC, WTLS ... In *EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 534–546. Springer, 2002.

[27] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, volume 2 of *WOEC*, pages 29–40. USENIX Association, 1996.