Threat for the Secure Remote Password Protocol and a Leak in Apple's Cryptographic Library

Andy Russon

Threat for the Secure Remote Password Protocol and a Leak in Apple's Cryptographic Library

Andy Russon[1,2]

[1] Orange, Applied Crypto Group, Cesson-Sévigné, France
[2] Univ Rennes, CNRS, IRMAR - UMR 6625, F-35000 Rennes, France
andy.russon@univ-rennes1.fr

Abstract. The Secure Remote Password protocol is a password-based authenticated key-exchange between two parties. One advantage is to prevent offline dictionary attacks from an adversary eavesdropping the communication. We present how such an attack is feasible if the modular exponentiation at the heart of the protocol is vulnerable and leaks some data related to the password.
In the case of a fixed exponent, adding randomness during the execution is a classical protection mechanism, and such a mechanism is present in Apple's cryptographic library to randomize the exponent. Despite being intended to protect against complex side-channel attacks, we show that its usage makes the implementation vulnerable to simple side-channels such as power analysis.
This leakage observed in the library is mild but is useful for the attack we propose on the Secure Remote Password protocol.

Keywords: Apple · Dictionary attack · Euclidean splitting · Exponentiation· Secure Remote Password

1 Introduction

The Secure Remote Password (SRP) protocol [21,24] allows two parties to securely establish a session key. It belongs to the class of augmented Password-based Authenticated Key-Exchange (PAKE), attributing an asymmetric role to the two parties. One is a client and knows a secret password, and the other is a server that only stores a verifier of the password. The advantages over a Diffie-Hellman key-exchange protocol are that its design ensures mutual authentication of the parties and protection against a Man in the Middle that eavesdrops or even interferes with the communication between the client and the server.

Calculations in the SRP protocol require several modular exponentiations in a finite field defined by a large prime. The exponents are secret, and one of them is specifically derived from the password. The knowledge of this value is sufficient to impersonate a client. An open question is whether partial knowledge of it could be enough.

Recently, it has been shown that implementations of a similar PAKE protocol can lead to a timing difference related to the password [4,22]. The authors used this leak as a distinguisher to filter passwords in a dictionary that do not correspond. While it is not a secret exponent that is derived from the password, a similar approach could be applied to the SRP protocol if the exponentiation algorithm is vulnerable to side-channels.

Many methods have been proposed to attack implementations of exponentiation algorithms. One category of those is passive attacks, consisting of observing the executions [15], either remotely by measuring the time taken for the calculation, or with physical access using probes to capture power consumption or electromagnetic emanations. Targets of these attacks are mainly algorithms whose behavior is dependent on the secret exponent, revealing the whole exponent or partial knowledge of it. These attacks remain relevant as recent research shows [1,2,14].

To protect against these side-channel attacks, several methods have been designed over the years. Timing attacks and simple power analysis can be avoided using algorithms with regular behavior to assure a constant-time flow of operations independent of the secret exponent. One example is the Montgomery ladder algorithm [17] that processes each bit of the exponent with the same operations. In the case of a fixed exponent, such as the SRP protocol, the exponentiation is vulnerable to more complex attacks that require the capture of many traces of execution [16]. Mechanisms relying on randomization were introduced as a protection against those attacks. One of the most known is exponent blinding by adding a random multiple of the order of the base element [9]. Another one central to this paper consists of splitting the exponent randomly in several shares [7].

However, these countermeasures do not necessarily protect from all attacks. For instance, the blinding of the exponent on NIST elliptic curves might not mask the exponent entirely due to the particular nature of the sparse order of the curves [10,20]. Concerning the exponent splitting, it was shown a correlation between the shares in the additive version of the splitting [18], and a template attack was applied on the Euclidean version of the splitting [12].

It is of interest to look at the choices made to achieve security of exponentiation algorithms in widely used cryptographic libraries, in particular for use with the SRP protocol. The source code of Apple's cryptographic library is made available to allow for "verification of its security characteristics and correct functioning".[1] This library implements the low-level implementations of primitives that can be used by developers through high-level APIs for security operation for operating systems iOS and macOS, in particular through the CommonCrypto interface for cryptographic operations or the Security Framework. Several protection mechanisms are implemented in the modular exponentiation used in the SRP protocol, including the Euclidean splitting technique, and analysis has shown the presence of a variation in its execution.

[1] https://developer.apple.com/security/ (bottom of the page).

In this paper, we introduce an attack on the SRP protocol when the underlying exponentiation algorithm is vulnerable to side-channels. This allows running an offline dictionary attack with two variants. The first where the attacker only observes the communication and the vulnerable exponentiation, while the other assumes an attacker that interferes as a Man in the Middle similar to the Dragonblood attack on the Dragonfly handshake [22]. We also present a vulnerability in the modular exponentiation of Apple's cryptographic library. We found that randomization of the exponent with the Euclidean splitting technique of [7] has been added in the most recent version of the library.[2] We show that it leads to a small leakage that can be measured with simple side-channels. While it is mild, it becomes much more significant from many measurements with a fixed exponent, making it possible to approximate the exponent by placing it in a smaller range. This is useful for the attack on SRP, in particular for the first variant. Although not present in the core of the paper, we found the same vulnerability in the exponentiation algorithm used with elliptic curves in the library.

The paper is organized as follows. We start in Sect. 2 with a general description of the SRP protocol. We introduce in Sect. 3 our attack on the protocol in the situation where the client uses a vulnerable exponentiation algorithm, even in the case of a small leak. Section 4 describes the modular exponentiation in Apple's cryptographic library and the leakage we found due to the Euclidean splitting of the exponent that leads to an approximation of the secret exponent. In Sect. 5, we present a simulation of a power trace of the modular exponentiation, and experiments to illustrate the effectiveness of the attack on SRP using the leak from Apple's library. We present countermeasures to avoid the leak with the Euclidean splitting in Sect. 6, and a conclusion in Sect. 7.

Responsible Disclosure. The vulnerability has been disclosed to Apple Product Security following their procedure. A security update has been made available in iOS 14.5 and macOS 11.3.

2 The Secure Remote Password Protocol

The Secure Remote Password (SRP) was introduced in [25] and is described as SRP-3 in RFC 2945 [24], and SRP-6 for use in TLS authentication is described in RFC 5054 [21]. The difference between the two versions is minor and addresses vulnerabilities that are not relevant to the paper, and we will describe the protocol using versions SRP-6 and SRP-6a. It is a password-based protocol whose main goal is to establish a key agreement between two parties in a client/server model. The password is known only from the client, while the server stores a *verifier*. They authenticate themselves by sending ephemeral data similar to a Diffie-Hellman exchange. As a result of successful authentication, the two parties share a cryptographically strong secret.

[2] No version number is indicated, but copyright notice and last file update refer to late 2019.

The protocol is designed such that the password cannot be derived from the communication between the parties. Therefore, an eavesdropper is unable to run an offline dictionary attack, making the protocol secure against weak passwords (e.g., a 4 or 6-digit passcode). Furthermore, if the server is compromised, the leaked data is insufficient to impersonate the client if the password is strong enough.

The protocol is described below and summarized in Fig. 1.

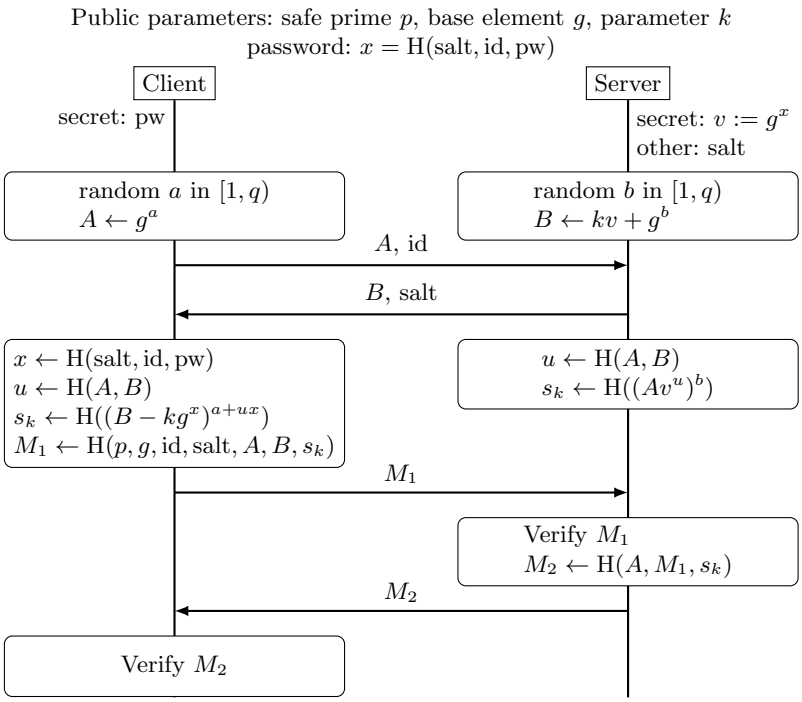

Fig. 1. Secure Remote Password (SRP) protocol, version SRP-6(a).

2.1 Description

Initialization. All calculations are performed in a finite field defined by a large prime p, and a base element g that generates a large multiplicative subgroup of order q. In particular, it is recommended that p is a safe prime (i.e., $p = 2q + 1$ with q a prime) for maximal security. Such parameters are defined in [21] to be used with the protocol for TLS authentication.

Before any authentication, a password **pw** and a salt must be chosen by the client, and a secret exponent x is derived as follows:

$$x = \mathrm{H}(\mathrm{salt} \mid \mathrm{H}(\mathrm{id} \mid \text{“:”} \mid \mathrm{pw})),$$

where H is a secure hash function and | is a concatenation. For simplicity, we will omit the specific construction of the input of the hash function and specify only the values that influence the output. The client computes $v := g^x \bmod p$, and the server stores the verifier v and the salt. The client does not need to store the value x. The matter of how the server gets the verifier and the salt depends on the application that uses the protocol and is not a part of the description.

Start of Authentication. The first phase consists of the client that sends their identity and a public value $A := g^a \bmod p$ where a is an ephemeral exponent randomly generated. From the client's identity, the server retrieves the corresponding verifier and salt, then sends the salt and a public value $B := kv + g^b$ where b is an ephemeral exponent randomly generated, and k a public parameter of the protocol ($k = 3$ in SRP-6, while it is deterministically generated from the other parameters in the SRP-6a variant).

Challenge Processing and Key Agreement. The two parties can compute a value u derived from the public outputs A and B. Then we have the following equality:

$$(B - kg^x)^{a+ux} \bmod p = (Av^u)^b \bmod p.$$

Using the salt to calculate x, the client can compute the expression on the left, and the server the one on the right. From this shared secret, a session key s_k is derived. A final step is necessary to prove to each other that their keys are identical. Both client and server exchange a value that each can verify. In case of a disagreement, the authentication is aborted.

2.2 Security

To impersonate a client, it is necessary to learn the exponent x for the challenge processing phase and construct a shared secret with the server. This value is neither exchanged nor stored, and cannot be derived from the public outputs exchanged. In the case the server is compromised, an attacker learns $v := g^x \bmod p$ and x can be recovered only by dictionary attack if the password is weak (the attacker finds candidates x' for x until $g^{x'} \bmod p$ is equal to v), or by solving a discrete logarithm problem. The latter is a hard problem if the group parameters are secure (the record as of the writing of this paper is a discrete logarithm in a 795-bit finite field [3]). This is equivalent to a breach of database password hashes and is not in the security scope of the SRP protocol.

On the client's side, the value x is reconstructed when the client enters the password, using the salt sent by the server, and needs to compute the exponentiation $g^x \bmod p$. Therefore, this operation is susceptible to be done several times and is the main topic of the paper.

3 Attack on the SRP Protocol

In this section, we present the attack on the SRP protocol. The goal is to run an offline dictionary attack against weak passwords using partial knowledge of the secret exponent obtained from a vulnerable exponentiation. While everything is described using the SRP protocol, we point out that the attack could be adapted for similar protocols involving an exponentiation related to a password.

3.1 Attack Model

The target of the attack is the client's side implementation of the protocol on the part relevant to the exponentiation with the secret exponent x derived from the password. The attack consists of three main steps:

1. Obtention of the salt and client's identity (once for each salt);
2. Observe through side-channels the vulnerable implementation (might be necessary to repeat this step depending on the vulnerability);
3. Run a dictionary attack using the side-channel leakage.

The first step is necessary since those values are the other entries outside of the password for the exponent derivation. The second allows the attacker to collect data related to the exponent x that is leaked by the implementation.

When those conditions are met, the attacker creates a distinguisher of the secret exponent from the side-channel leakage. Then, from the salt and the client's identity previously obtained, a dictionary attack can be performed as summarized in Algorithm 1. The distinguisher acts as an indicator to filter out wrong passwords. Indeed this value is derived from the correct exponent, so the correct password will satisfy the verification against the distinguisher. However, it does not mean the correct password has been found, and unless great precision, many other passwords will pass the test. The performance of filtering out wrong passwords depends heavily on the precision given by the distinguisher.

Algorithm 1 Password filtering given a distinguisher.

Require: salt, id, dictionary, distinguisher
Ensure: list of password candidates
1: list $\leftarrow \{\}$
2: **for** pw' **in** dictionary **do**
3: $\quad x' \leftarrow \mathrm{H}(\text{salt}, \text{id}, \text{pw'})$
4: $\quad$ **if** check$(x', \text{distinguisher})$ **then**
5: $\quad\quad$ list $\leftarrow$ list $\cup \{\text{pw'}\}$
6: **return** list

3.2 Passive Attacker

In this version, the attacker only observes the execution on the client's side. From the observation of one or several executions, a single distinguisher is created. The password filtering is limited when the leak is identical in each execution. The main problem being one cannot improve the distinguisher precision with more observations. Nonetheless, it is more interesting with a vulnerability that leaks more data at each execution, as is the case in Apple's cryptographic library.

One possibility to make this attack more effective is to compare the list of password candidates of two users. If the number of password candidates in each list is low enough, only a few wrong passwords will appear on both lists. Then, a common password can be found in the intersection of the two lists.

3.3 Active Attacker

The previous attacker is limited to one distinguisher and its precision. In the Dragonblood attack [22], it was noticed that the modification of a MAC address allowed the acquisition of fresh measurements from an identical password, i.e., a new distinguisher. The same idea can be applied in the SRP protocol using the salt which influences the output of the hash function. An alteration of this value with a Man in the Middle attack results in a new exponent derived from the same password. A distinct distinguisher can be created from the side-channel leakage, and the password filtering of Algorithm 1 can be used to reduce even further the number of candidates iteratively.

This variant implies that the session key computed by the client is incorrect, and this could be detected with many attempts. To prevent detection on the server's side, an alternative would be an attacker that poses as the server and sends directly the chosen value for the salt. It would still imply failed authentication for the client.

Another possibility would be a service or product that changes the salt value for an identical password.

3.4 Practical Considerations

Obtention of the Client's Identity and Salt. The client's identity (email address, username, identification number, etc) can be easily guessed when the target is identified. As for the salt, it is exposed to an eavesdropper when the communications are sent in plaintext.

However, the communication could be encrypted using a certificate-based TLS, preventing the exposure of the salt. A bypass of this added security is possible if the attacker initializes an authentication with the server by posing as the client. The attacker would receive an ephemeral public output B and the salt that corresponds to the client. An accessible example is the ProtonMail service that uses SRP for authentication since version 3.6 [19]. The data of the SRP protocol are exchanged in JSON format and can be extracted using a web

browser inspector. Then it is easy to get the salt of any user by attempting an authentication.

Another layer of security could be present in the application that uses SRP, such as a second factor of authentication before the start of the SRP protocol. This would make it more difficult to pose as a client and retrieve the salt. We give details in Sect. 3.5 of the case of the Apple iCloud Keychain recovery service.

Side-Channel Observation. A leakage that can be observed at distance such as timing attacks is difficult to obtain in this protocol: calculations involving the secret exponent are only a part of the whole process, so timing analysis could be hard or impossible to exploit. Other means of attacks require physical access to the client's device, whether it is a smartphone or a personal computer. This is a limit to the application of the attack since the observation follows the entering of a password.

However, the authors of [11] have shown how a side-channel attack can be mounted using a magnetic probe near a device, or a power probe on the phone's USB charger in a discreet manner. In particular, their experiments were on an Apple iPhone.

Man in the Middle. The main problem for the second variant of the attack is that certificate-based TLS communication between client and server may harden the possibility to modify transmitted values such as the salt. So the Man in the Middle attacker would require to impersonate the server. A salt modification implies a failed authentication, and the client might take this as a mistyped password. Since it only needs to be done a few times, the attacker can space out the attack over time to pass unnoticed.

There are other possible leads to make the second variant possible such as a fault injection in the salt or the user's identity on the client device. Those solutions come with their difficulties; the one of importance here is that the effect must be controlled so the modified salt (or user's identity) must be predictable by the attacker.

Distinguishers and Efficiency of Dictionary Attack. To estimate the efficiency of the dictionary attack, we illustrate with the distinguisher that the paper focuses on: the exponentiation leaks data revealing that the exponent lies in a small interval of width w. The smaller this value is, the more effective the filtering.

The explanation lies in the construction of the exponent in the SRP protocol where it is the integer representation of the output of a cryptographic hash function. Let ℓ the maximum bit length possible of a hash function (e.g., $\ell = 256$ for SHA-256), then their outputs are expected to be evenly distributed so the exponent can be any integer in the interval $[0, 2^\ell - 1]$. Under the same salt and user's identity, running through all passwords in a dictionary will result in exponents uniformly distributed, so around $w/2^\ell$ of the exponents are expected to

satisfy the distinguisher. If D is the dictionary size, then the number of password candidates is close to

$$D \cdot \frac{w}{2^\ell}.$$

In the context of the paper, this distinguisher is made possible by a leak explained in Sect. 4.1, and present in Apple's cryptographic library. The width w of the interval is related to the number of measurements of the exponentiation. That makes the first variant of the attack interesting in practice since the only lever to reduce the number of password candidates is to refine the distinguisher.

However, the use of several distinguishers in the second variant makes the filtering very effective. Indeed, a modification of the salt implies a fresh exponent and distinguisher that are independent of the original values thanks to the hash function. Then the list of password candidates can be reduced significantly by each distinguisher of width w_i:

$$D \cdot \prod_i \frac{w_i}{2^\ell}.$$

In particular, the case where two users share the same password is equivalent to the possession of two distinguishers of widths w_1 and w_2.

If the distinguisher is the bit length n of the exponent, then the exponent lies in an interval of width $w = 2^{n-1}$. Thus, around $1/2^{\ell-n+1}$ of the passwords correspond, and in the worst case, it means half the passwords are expected to be filtered out. A few dozens bit length distinguishers are generally enough depending on the dictionary size. We refer to the Dragonblood paper [22] for this particular case. While not related to the bit length of an exponent, the leak also follows a geometric distribution.

Remark 1. The reliability of the distinguisher is important. For instance, it is not guaranteed that the exponent lies in the subinterval obtained with the analysis of the leak in the modular exponentiation in Apple's cryptographic library, which could filter out the correct password.

3.5 iCloud Keychain Recovery

The iCloud ecosystem is the heart of Apple's online services and can be shared across devices from one account, and the SRP protocol is used in several places. One of the services is iCloud Keychain Recovery that allows users to escrow their keychain with Apple (that contains sensitive data such as passwords or credit cards). This service has a supplementary layer of protection through the SRP protocol (with the 2048-bit group of RFC 5054, SHA-256 as the hash function, and 64 bytes salts) using a separate password from the iCloud account: for each device, iCloud stores a backup of the keychain as a record protected with the device's password (the default when the iCloud account has a second factor of authentication activated). Thus, the keychain is secured if the iCloud account is

compromised, and the content stays inaccessible from Apple. More details on the iCloud Keychain services are given in the Apple Platform Security guide.[3]

Our investigation has shown the following HTTP requests when a user signs out and back into iCloud on their device:

1. A request `get_records` is transmitted, and the server answers with a list of records associated with the iCloud account;
2. A request `srp_init` initiates the first phase of the SRP protocol containing the client's ephemeral value to access a record, and the server answers with the corresponding data: `DSID` (unique identifier of the iCloud account used as identity in SRP), the server's ephemeral value, and the salt;
3. A request `recover` for the second phase with the client's proof, and the servers replies with its proof and the record;
4. The client sends a request `enroll` with a new record protected with the same password, but with a different salt.

We have seen two situations where the recovered record is either the old one corresponding to the device or one corresponding to another enrolled device. The first case is interesting for the variant of the attack that uses several distinguishers.

Though, the realization of the attack requires the obtention of the salt and identifier. Monitoring the encrypted communications between the target's device and Apple's servers could overcome this issue, but our attempts by setting up a proxy server failed on a MacBook Pro. An alternative would be to run the `srp_init` request, but a password-based token is necessary on 2FA protected iCloud accounts (which has become mandatory for new accounts and cannot be deactivated). This token is acquired after successful authentication on the iCloud account.

In the case an attacker has already compromised an iCloud account and can bypass the second factor of authentication once, then the device can be enrolled as a trusted device. By doing so, it might happen the keychain will be synchronized directly with the other enrolled devices if the keychain option is checked in system preferences. This synchronization does not use SRP, so that makes our attack useless if the goal is to retrieve a keychain, but it could still be used to recover the target's device password.

The attack to recover a weak password on an Apple device can be realized under the following assumptions:

- Compromise the iCloud password of the targeted user;
- Bypass the second factor of authentication at least once;
- Force the user to disconnect their iCloud account on their device several times;
- Observe the side-channel leakage when the user reconnects during the execution of the SRP protocol in iCloud Keychain recovery.

Every time a new leak and salt are obtained, the dictionary attack can be performed as given in Algorithm 1.

[3] https://support.apple.com/guide/security/welcome/web.

4 Modular Exponentiation in Apple's Library

This section presents the modular exponentiation in Apple's cryptographic library and the leakage from the exponent randomization with the Euclidean splitting technique.

4.1 Exponent Randomization

Let x an exponent of n bits whose binary representation is

$$x = \sum_{i=0}^{n-1} x_i 2^i.$$

This value is randomized by choosing a random integer m of λ bits and compute the Euclidean division of x by m. The quotient is $a = \lfloor x/m \rfloor$ and the remainder is $b = x \bmod m$, then the exponentiation g^x is rewritten as

$$g^x = (g^a)^m g^b,$$

with three exponentiations.

This technique is called the Euclidean splitting and was introduced in [7] as an alternative to other exponent blinding methods. The authors proposed to compute simultaneously the exponentiation with the quotient a and the remainder b with Strauss-Shamir double exponentiation [8, Algorithm 9.23] for efficiency. In the case of Apple, it is computed as three successive individual exponentiations.

Quotient Bit Length Variation. A variation regarding the quotient bit length and related to the exponent was found: when divided by an integer of a fixed bit length, then the quotient bit length has two possible values. The probability that each of them is produced depends on the position of the exponent in the interval $[2^{n-1}, 2^n)$.

The definition and theorem below give the details on the partitioning of the interval and the associated probabilities.

Definition 1. *Let n, λ and β three non-negative integers with $\lambda \leq n$ and $\beta \in [0, 2^{\lambda-1})$. We note*

$$I(n, \lambda, \beta) = \big[\, 2^{n-\lambda}(2^{\lambda-1} + \beta), \quad 2^{n-\lambda}(2^{\lambda-1} + \beta + 1) \,\big),$$

a subinterval of $[2^{n-1}, 2^n)$ of width $2^{n-\lambda}$.

Theorem 1. *Let d the bit length of the quotient of the Euclidean division of an integer x by an integer m chosen uniformly at random in $[2^{\lambda-1}, 2^\lambda)$. Then d is either $n - \lambda$ or $n - \lambda + 1$ and we have*

$$\Pr[\, d = n - \lambda + 1 \mid x \in I(n, \lambda, \beta) \,] = \frac{\beta + 1}{2^{\lambda-1}}. \tag{1}$$

Proof. Suppose $x \in I(n, \lambda, \beta)$ and let $x = am + b$ the Euclidean division of x by an integer m of λ bits. The lower and upper bounds on the quotient a are

$$\frac{2^{n-1} - b}{2^{\lambda}} \leq a = \frac{x - b}{m} < \frac{2^{n} - b}{2^{\lambda-1}},$$

and since $b < 2^{\lambda}$ and a is an integer, we have $a \geq 2^{n-\lambda-1}$. Overall, only the bit length $n - \lambda$ and $n - \lambda + 1$ are possible.

Now, if $m \leq 2^{\lambda-1} + \beta$, then we have

$$a = \frac{x - b}{m} \geq \frac{2^{n-\lambda}(2^{\lambda-1} + \beta) - b}{m} \geq 2^{n-\lambda} - \frac{b}{m},$$

and since $b < m$ and a is an integer, we have $a \geq 2^{n-\lambda}$ so the bit length of a is $n - \lambda + 1$. In this case there are $\beta + 1$ possible values for m out of $2^{\lambda-1}$, hence the probability.

The other case is when $m > 2^{\lambda-1} + \beta$, then we have

$$a < \frac{2^{n-\lambda}(2^{\lambda-1} + \beta + 1) - b}{m} \leq 2^{n-\lambda},$$

and the bit length of a is $n - \lambda$. □

The consequence is that if m is randomly generated at uniform and its bit length is λ, then the probability that the quotient bit length is $n - \lambda + 1$ depends on which interval $I(n, \lambda, \beta)$ contains the exponent x. It is represented in Fig. 2 for $\lambda = 4$ as an illustration, to make the staircase apparent, and $\lambda = 32$ the use case in Apple's library, where the intervals are too small and close to a linear correlation between an exponent and the probability.

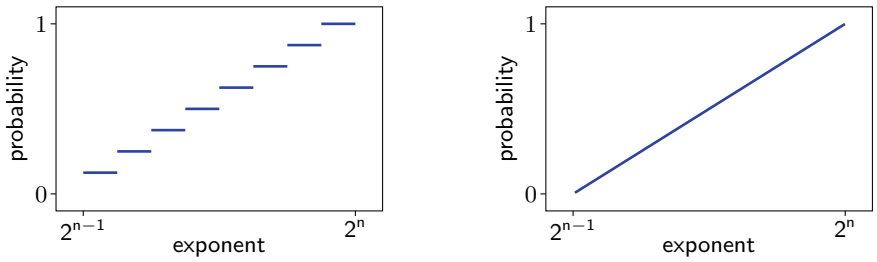

Fig. 2. Correlation between an exponent of n bits, and the probability that the bit length of the quotient of the Euclidean division with a λ-bit divisor is $(n - \lambda + 1)$ (left: $\lambda = 4$, right: $\lambda = 32$).

Exponent Approximation. An oracle that reveals $d = \lceil \log_2(x/m) \rceil$ for a fixed unknown integer x and N random integers m of an identical bit length λ follows a binomial distribution of parameters N and probability $(\beta+1)/2^{\lambda-1}$ according to Theorem 1. This oracle acts as binomial trials to construct a confidence interval on the unknown probability when only the number of repeated experiments N and the number of successes n_{obs} that d takes the value $n-\lambda+1$ are known. Since this probability is a characteristic of the interval $I(n, \lambda, \beta)$, then an approximation of x is deduced.

The steps are straightforward. A binomial proportion confidence interval $[p_1, p_2]$ on the probability is made observing the number of outcomes n_{obs} of successes (quotient bit length observed is $d = n - \lambda + 1$) out of N outcomes. Then, β can be estimated too by

$$p_1 2^{\lambda-1} - 1 \leq \beta \leq p_2 2^{\lambda-1} - 1,$$

and since β is an integer, let $\beta_{\min} = \lceil p_1 2^{\lambda-1} - 1 \rceil$ and $\beta_{\max} = \lfloor p_2 2^{\lambda-1} - 1 \rfloor$. Finally, the approximation on x is made by the concatenation of the contiguous intervals $I(n, \lambda, \beta_{\min})$ to $I(n, \lambda, \beta_{\max})$:

$$[2^{n-\lambda}(2^{\lambda-1} + \beta_{\min}), \quad 2^{n-\lambda}(2^{\lambda-1} + \beta_{\max} + 1)).$$

This interval is likely to contain the secret exponent x depending on the confidence level.

This approximation makes it possible to construct a distinguisher for the attack on the SRP protocol.

4.2 Exponentiation Algorithm

There are several exponentiation algorithms in the library. The blinded modular exponentiation is implemented in the function `ccdh_power_blinded`[4]. The overall process consists of three individual successive exponentiations, with other blinding techniques, and is summarized in Algorithm 2 with a random divisor of $\lambda = 32$ bits.

The exponentiation algorithm used from line 5 to line 7 is a 2-bit windowing method presented in Algorithm 3 called *square-square-multiply-always*. The number of iterations is dependent on the input bit length: if the input bit length is d, there are $\lceil d/2 \rceil$ iterations of the loop. This can be revealed by side-channels by counting the number of patterns on power consumption or electromagnetic trace (see Sect. 5.1). The divisor bit length is fixed and known in advance, but those of the quotient and the remainder are variable.

[4] In the file `ccdh/src/ccdh_power_blinded.c`.

Algorithm 2 Blinded modular exponentiation in Apple's CoreCrypto library

Require: x, g, p
Ensure: $g^x \bmod p$

Modulus, exponent, and base blinding
$p^\star \leftarrow$ **blinding**(p) ▷ $p^\star = p \cdot \text{random}$
$g^\star \leftarrow$ **blinding**(g) ▷ $g^\star = g + p \cdot \text{random}$
$m \leftarrow$ random integer in $[2^{31}, 2^{32})$
$a \leftarrow \lfloor x/m \rfloor$, $b \leftarrow x \bmod m$

Exponentiation
$t_1 \leftarrow {g^\star}^a \bmod p^\star$
$t_2 \leftarrow {t_1}^m \bmod p^\star$
$t_3 \leftarrow {g^\star}^b \bmod p^\star$
return $(t_2 \cdot t_3) \bmod p$ ▷ $(g + p \cdot \text{random})^{am+b} \bmod p = g^x \bmod p$

Algorithm 3 Square-square-multiply always exponentiation.

Require: g, $a = (a_{d-1}, \ldots, a_0)_2$ with $a_{d-1} = 1$
Ensure: g^a

for $i = 0$ **to** 3 **do**
    $\text{tab}[i] \leftarrow g^i$
$r \leftarrow 1$
for $i = \lceil d/2 \rceil - 1$ **down to** 0 **do**
    $r \leftarrow r^2$
    $r \leftarrow r^2$
    $r \leftarrow r \cdot \text{tab}[2a_{2i+1} + a_{2i}]$
return r

For a fixed exponent of bit length n, we have seen that there are two possible bit lengths d for the quotient and can be used to make an approximation on the exponent. Both can happen using different values for m, but it is not always possible to distinguish them with the *square-square-multiply-always*. If $n - \lambda$ is odd, then

$$\left\lceil \frac{n-\lambda}{2} \right\rceil = \left\lceil \frac{n-\lambda+1}{2} \right\rceil = \frac{n-\lambda+1}{2}, \tag{2}$$

so the quotient bit length is not leaked, but n can still be deduced from the formula. On the contrary, if $n - \lambda$ is even, then

$$\left\lceil \frac{n-\lambda+1}{2} \right\rceil = \left\lceil \frac{n-\lambda}{2} \right\rceil + 1, \tag{3}$$

so the algorithm runs with a different number of iterations for each of the possible quotient bit lengths, and the bit length n of the exponent x can also be deduced. Though, it requires several observations of exponentiations to be sure if we are situated in one or the other case. As a consequence, one measure is not sufficient to know exactly the bit length n, but the remark below can make it possible in the SRP protocol.

Remark 2. On the client's side in the SRP protocol, the exponentiation involving the exponent derived from the password is computed in the call of the function in line 4 of Fig. 3 using the blinded exponentiation. But we noticed it is also computed with a different algorithm in line 1, but the result stored in the variable v is never used thereafter (only to be cleared from memory). This algorithm is Montgomery ladder [17] and starts the process at the most significant bit, leaking the bit length of x. Therefore, the bit length can be known from one measure.

```
1 cczp_power(ccsrp_ctx_zp(srp), v, ccsrp_ctx_gp_g(srp), x)
      ;
2
3 /* Client Side S = (B - k*(g^x)) ^ (a + ux) */
4 ccsrp_generate_client_S(srp, S, k, x, u, B);
```

Fig. 3. Dummy exponentiation $g^x \bmod p$ on the client side in the SRP protocol in Apple's cryptographic library.

5 Experimental Results

In this section, we present the captures of power consumption of the modular exponentiation to observe the leak from Apple's library, and experiments to illustrate the effectiveness of the attack on SRP with this leak.

5.1 Power Trace Capture

This part has been made in collaboration with Cyril Delétré, a member of the author's team.

The variation of the quotient bit length can be revealed by side-channel by counting the number of patterns on power consumption or electromagnetic trace. We have selected the Raspberry Pi Zero SoC as it offers a good compromise between computing performances, hardware complexity, and power supply configuration. First, to get a constant computing capacity and minimize the noise on the power line the CPU frequency has been fixed to 700 MHz and the HDMI/TV output has been turned off (this allowed cleaner traces). Starting with a fresh Raspberry Pi OS setup we have built and installed the CoreCrypto library from the sources. Then we have written in C language a program that toggles a GPIO line on the SoC before and after the function from CoreCrypto has been called. This way the oscilloscope can:

- Measure the power consumption with a first probe connected to a resistor (3.3 Ohms in our case) in series on the 5V power line;
- Trigger the start of the computing on a second probe connected to the GPIO line (Figs. 4 and 5).

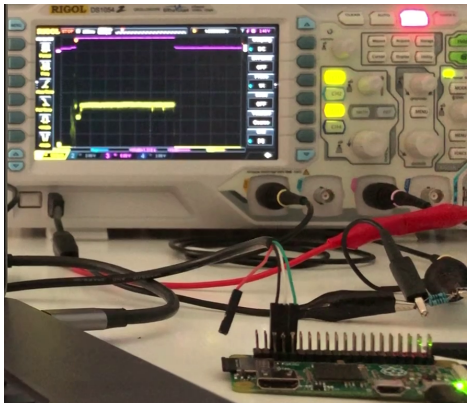

Fig. 4. Measurement of power consumption on a Raspberry Pi Zero.

Finally, we have automated the batch of measurements with a script written in Python 3 that:

- Launches the calculation on the Raspberry Pi Zero using the UART serial console (less disturbance compared to a USB or SSH/telnet console);
- Downloads after each run the data from the oscilloscope using its TCP remote console.

We chose the 2048-bit group of RFC 5054 [21] for the experiment, and the exponent

$$x = \begin{array}{l}\texttt{d3afa905fededc64bc907b809da3dcb} \\ \texttt{484763c25c3b4728bb081a97cf9f0a5}\end{array} \tag{4}$$

in hexadecimal format. For each execution, the pseudo-random number generator of CoreCrypto used in the function `ccdh_power_blinded` was initialized with a random seed and was repeated 10000 times.

We give two sample traces in Fig. 6 where we see that the major part of each trace corresponds to the exponentiation with the quotient (a vertical line indicates the beginning of the individual exponentiations), and a zoom on the end of this part in Fig. 7 reveals that it is shorter on the first trace than the second one by one pattern of *square-square-multiply*.

It is interesting to note that we do not necessarily need to count the patterns *square-square-multiply* every time since the approximation only needs to distinguish between the two cases. However, it is important to do it at least once to find the bit length of the exponent. In this example, we found that there are respectively 108 and 109 loop iterations on the quotient exponentiation. According to Eq. (3), we deduce that the exponent is a 248-bit integer which is consistent with the given value in Eq. (4).

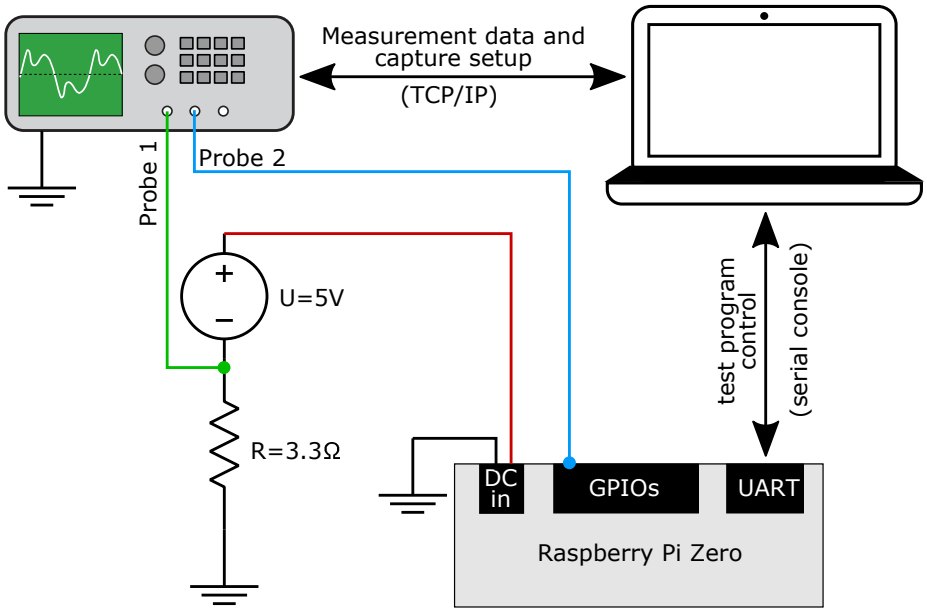

Fig. 5. Configuration of the experimental setup of the oscilloscope and Raspberry Pi Zero.

We remark that this can still be found even when the patterns are hard to distinguish if the noise is too high, as long as the beginnings of the individual exponentiations are exposed. Indeed, using the fact that the random divisor is always a 32-bit integer, then there are always 16 iterations of the loop. Therefore, the number of iterations with the exponentiation with the quotient can be deduced from the length between the two first dashed lines (taking into account the square and multiply of the precomputation).

We can find easily the start of the exponentiations in the captures thanks to the presence of a valley that corresponds to the first loop of the *square-square-multiply-always* algorithm. Because the exponentiation is initialized with the value $p - 1$, the second squaring is 1^2 followed by a multiplication by 1. Both operations manipulate very low Hamming weight values, and this has a significant impact on power consumption.

We used this to classify the traces in the two expected groups using the position of the valley corresponding to the first loop of the exponentiation with the random divisor. Many traces had major disruptions, but this method proved to work well enough for 9356 traces, with 6099 corresponding to the "109" group. Then the observed frequency was 0.6519 which is close to the probability we tried to estimate around 0.6538 for the exponent x.

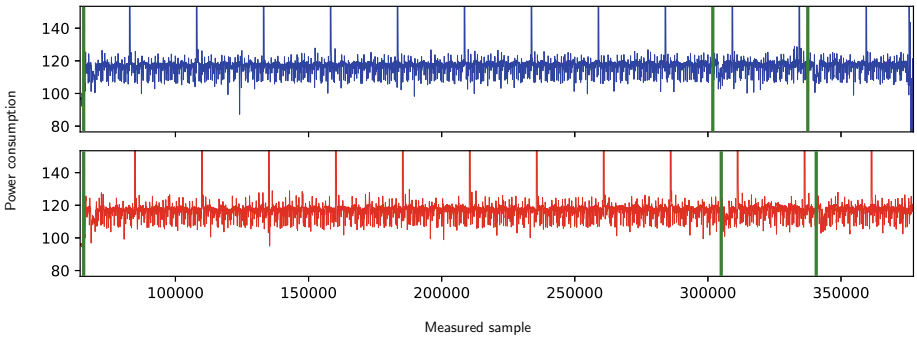

Fig. 6. Two power trace captures of an exponentiation blinded with the Euclidean splitting with an identical exponent (vertical lines represent the start of each of the three exponentiations).

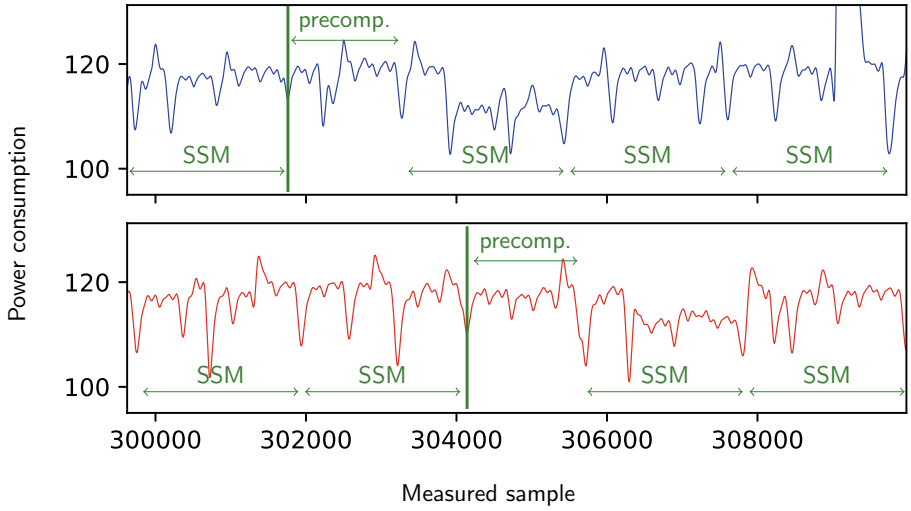

Fig. 7. Zoom on the end of the exponentiation with the quotient, and the start of the one with the random divisor. The second trace has one more pattern of *square-square-multiply* (SSM).

5.2 Dictionary Attack on SRP

The case of several bit length distinguishers known to the attacker has been well established in [4,22], so we focus on the distinguisher given by the Euclidean splitting of the exponent as implemented in Apple's library.

We ran the experiment with the following parameters for the SRP protocol:

- SHA-256 as the hash function (exponents are in the interval $[0, 2^{256})$);
- 2048-bit group of RFC 5054;
- Salt of 16 bytes;
- 6-digit password.

Given a random password and salt, the secret exponent x was derived according to the SRP protocol (using "id" as the client's identity). We simulated the leak for $N = 1000$ exponentiations following the description of the modular exponentiation in Apple's library, then an approximation was made in the form $[x_{\min}, x_{\max}]$ using the Wilson score interval [23] and a confidence level of 95%, as described in Sect. 4.1. It has the advantage over a normal approximation to be more suited for a small sample size N or when the probability to estimate is close to 0 or 1. All guessed passwords whose derived exponent fell in the interval were kept as candidates.

We repeated the experiment for 10000 different passwords and collected the exponent values, the number of password candidates found, and if the password was correctly included amongst the candidates. The results are given in Fig. 8.

An approximation on the exponents of odd bit length cannot be made as we have seen in Sect. 4.2, so the number of password candidates for those is way higher and does not appear (except for exponents of odd bit length less than 251 represented by the isolated dots on the left).

As a result, there were only 6846 cases with less than 32000 password candidates, and the correct password was included in the list for 95.5% of them, which is consistent with the confidence level.

We see that the number of candidates is way lower for exponents near a power of 2. This is a consequence of the Wilson score interval that gives a better approximation when the probabilities are near 0 or 1. Therefore there is a concave curve between each power of 2 for exponents of even bit length.

The best result was a password with a corresponding exponent around $2^{243.17}$ and was successfully included amongst a list of 8 candidates.

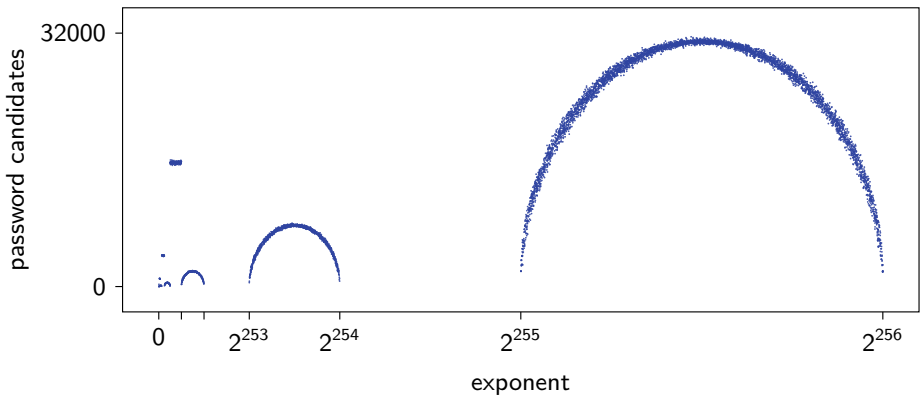

Fig. 8. Number of password candidates with $N = 1000$ measures as a function of the secret exponent x.

In a second experiment, we looked at the effects of the number N of measures on the number of password candidates. We kept the same settings as the previous with the two following changes:

- Exponents have bit length $n = 256$;
- The number of measures N ranged from 10^2 to 10^5.

The results are given in Fig. 9, where we can see that when the sample size is increased by a factor of 10, the number of password candidates is decreased by a factor around $\sqrt{10}$ due to the binomial approximation.

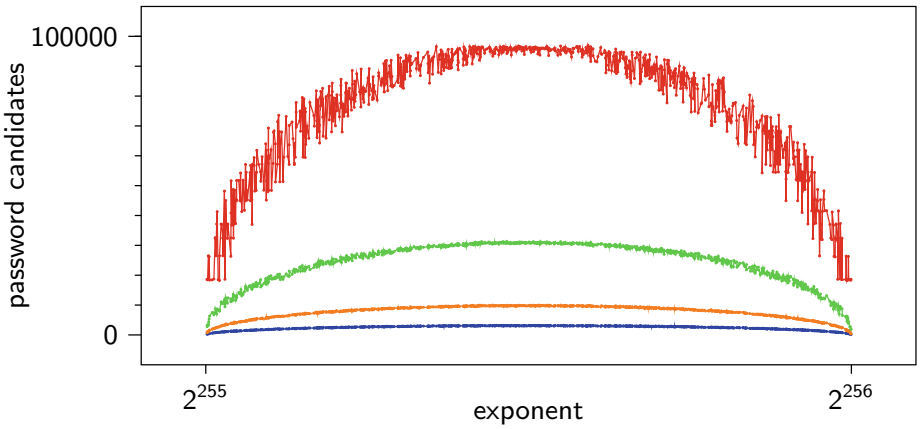

Fig. 9. Number of password candidates as a function of the secret exponent (from top to bottom: $N = 100$, $N = 1000$, $N = 10000$, and $N = 100000$).

6 Countermeasures

In this section, we present our proposition to make the Euclidean splitting protected against our attack, followed by the countermeasure proposed by Apple developers.

6.1 Our Proposition

It is necessary to hide the bit length of the exponent to prevent the attack on SRP using a bit length distinguisher. This can be done using the padding proposed in [5] where x is padded with the group order q so the result of the exponentiation is unchanged. Combined with an exponentiation algorithm with a regular behavior, nothing on the exponent is leaked from simple side-channels.

However, in the context of SRP, the group order can be much higher than the exponents, so the above technique adds a large cost to the execution. We

present an alternative using a precomputed value. If the exponent x can have a maximum bit length ℓ, then it can be replaced by

$$x_{\text{pad}} = x + 2^{\ell}.$$

Let $y := g^{-2^{\ell}} \bmod p$ a precomputed value, then $g^x \bmod p$ can be calculated as follows:

$$g^{x_{\text{pad}}} \cdot y \bmod p.$$

The extra cost is moderate since it adds one multiplication, and the exponentiation involves an exponent with one bit longer than the largest exponent possible.

The Euclidean splitting can be used, but it is necessary to take care of the quotient bit length. We suppose the exponent has been replaced by the padded value x_{pad} with one of the two techniques above, and the Euclidean splitting technique is applied with a random divisor m of λ bits:

$$a = \lfloor x_{\text{pad}}/m \rfloor, \quad b = x_{\text{pad}} \bmod m.$$

From the padding technique we know the bit length n of x_{pad} in advance, so the two possible quotient bit lengths are $n - \lambda$ and $n - \lambda + 1$. We can replace the quotient a by

$$a_{\text{pad}} = a + 2^{n-\lambda+1}$$

to hide the bit length, and use the precomputed value $z := g^{\lambda-n-1}$ to compute $g^{x_{\text{pad}}} \bmod p$ as:

$$(g^{a_{\text{pad}}} \cdot z)^m \cdot g^b \bmod p.$$

Again, the extra cost is moderate since it adds one multiplication and only 1 or 2 bits on one exponent.

Remark 3. In the case where the exponentiation algorithm is the 2-bit windowing method of Algorithm 3 used in Apple's library, the quotient bit length can be directly hidden with a padding on x only. With exponents less than 2^{ℓ} for ℓ even, the padded value $x_{\text{pad}} = x + 2^{\ell}$ is an integer of odd bit length $\ell + 1$. In Sect. 4.2 we saw that the number of iterations of the loop with the quotient as an exponent is always the same in this situation (when λ is even).

6.2 Apple's Proposition

The solution retained by Apple is to replace the exponentiation by the Montgomery ladder algorithm [17] given in Fig. 10, while still using the Euclidean splitting.

The algorithm works even if the leading bits of the exponent are set to 0. However, this case would imply a multiplication by 1 and a squaring of 1 which is easy to detect on a power trace as we proved in Sect. 5.1. To avoid this pitfall, the modular arithmetic has been replaced with the Montgomery representation so the unity does not have a low Hamming weight representation. This is done by the function `cczp_to_ws` before the loop in Fig. 10.

When used for iCloud Keychain, we have confirmed that the loop bounds are fixed with 225, 32, and 32 loop iterations respectively for the exponentiations with the quotient, random divisor, and remainder.

```
ccn_set(n, r1, s);
ccn_seti(n, r, 1);
cczp_to_ws(ws, zp, r, r);

cc_unit ebit = 0;
for (int bit = (int)ebitlen - 1; bit >= 0; --bit) {
    ebit ^= ccn_bit(e, bit);
    ccn_cond_swap(n, ebit, r, r1);
    cczp_mul_ws(ws, zp, r1, r, r1);
    cczp_sqr_ws(ws, zp, r, r);
    ebit = ccn_bit(e, bit);
}

// Might have to swap again.
ccn_cond_swap(n, ebit, r, r1);
```

Fig. 10. Excerpt of the Montgomery ladder for modular exponentiation in the updated version of CoreCrypto.

7 Conclusion

We showed an attack on the SRP protocol when the modular exponentiation is vulnerable to side-channels, so an attacker can perform a dictionary attack to find a weak password. Then, we presented a leak we found in Apple's cryptographic library that comes from the presence of a protection mechanism that randomizes exponents through Euclidean divisions. We analyzed it and showed that a passive attacker can approximate the secret exponent from several measures, and run an offline dictionary attack on SRP. It is not negligible from a few thousand observations of the exponentiation and can produce a list of hundreds of candidates from a dictionary containing millions of passwords.

Our findings were shared with Apple in responsible disclosure and the vulnerability was patched in iOS 14.5 and macOS 11.3.

Like other works, this paper highlights that insecure exponentiations are still deployed in cryptographic libraries and that a small leak can be enough to diminish the security of a protocol such as SRP.

Acknowledgments. The author would like to thanks the anonymous reviewers for their comments, Apple Product Security for their collaboration, and finally his colleague Cyril Delétré who provided the power trace captures.

A SRP Requests and Responses in iCloud Keychain Recovery

In this appendix, we present the HTTP request `srp_init` in the context described in Sect. 3.5.

The captures were made on a secondary device using Wireshark[5] and Frida[6] to export the TLS session keys for decryption.

```
POST /escrowproxy/api/srp_init HTTP/1.1
Host: p49-escrowproxy.icloud.com:443
(...)
Authorization: Basic Y29 (...) BFVA==
(...)

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.
    com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
 <key>blob</key>
   <string>pIC8heH+SbClHjnugsfBBc (...) +2+CM8q8hItheOscqwA==</string>
 <key>command</key>
   <string>SRP_INIT</string>
 <key>label</key>
   <string>com.apple.icdp.record.et3n (...) 8HqM</string>
 (...)
</dict>
</plist>
```

Fig. 11. Client's side of the initialization phase of the SRP protocol in iCloud Keychain recovery.

In Fig. 11, the ephemeral value A is base64 encoded and corresponds to 256 bytes, consistent with the 2048-bit group. The server's answer is given in Fig. 12, and contains the salt and ephemeral value B at the end of the data in the base64 string `respBlob`, each preceded by their length in bytes: 64 and 256.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.
    com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
 <dict>
  <key>status</key>
    <string>0</string>
  <key>message</key>
    <string>Success</string>
  <key>version</key>
    <integer>1</integer>
  <key>dsid</key>
    <string>28 (...)</string>
  <key>ClubTypeID</key>
    <integer>0</integer>
  <key>respBlob</key>
    <string>AAABiAAAAKQAAAAAPbZQrX (...) jsxg48nknPybRNHkTM=</string>
 </dict>
</plist>
```

Fig. 12. Server's side of the initialization phase of the SRP protocol in iCloud Keychain recovery.

[5] https://www.wireshark.org
[6] https://frida.re

We confirmed on the first device that the SRP protocol is executed to retrieve the same record and salt after signing in and out. The new record created for the first device is the one that is used on the secondary device when the experiment is repeated.

B Elliptic Curve and SPAKE2+

In this appendix, we present briefly the implementation of the exponentiation (scalar multiplication) with elliptic curves where the Euclidean splitting technique is also used. We first present the differences of the algorithm implementation, and the consequence on the password-based authenticated key-exchange protocol SPAKE2+, which can be attacked similarly as with SRP.

B.1 Elliptic Curve Scalar Multiplication

The elliptic curves named `secp192r1`, `secp224r1`, `secp256r1`, `secp384r1`, and `secp521r1` share the same exponentiation algorithm that is implemented in the function `ccec_mult`.[7]

The whole exponentiation is given with generic group notations in Algorithm 4. It is randomized with the Euclidean splitting, but we note differences with the previous case of modular exponentiation:

- A padding to hide the bit length of the exponent x with the group order is applied [5];
- A padding is applied on the remainder of the Euclidean division, so the bit length of the remainder is hidden;
- The individual exponentiations are executed with the Montgomery ladder algorithm that leaks the bit length of the exponents.

With a padding on the remainder, the only variation in the execution of the `ccec_mult` function is the exponentiation with the quotient. As a consequence, the timing execution of the whole exponentiation leaks the bit length of the quotient. An approximation of the secret exponent can be done with timing analysis if the auxiliary processing before and after the call of the function can be controlled.

This issue has been addressed in the updated version of the library. The Montgomery ladder algorithm has been tweaked to make it work when leading bits are set to 0 using characteristics of the algorithm and the point addition formulas from co-Z arithmetic [13].

[7] In the file `ccec/src/ccec_mult.c`.

Algorithm 4 Blinded elliptic curve scalar multiplication with NIST elliptic curves in Apple's cryptographic library

Require: x, g of order q
Ensure: g^x

1: $x_1 \leftarrow x + q - 2^{32}$
2: $x_2 \leftarrow x + 2q - 2^{32}$
3: **if** $\lceil \log_2(x_1) \rceil = \lceil \log_2(q) \rceil + 1$ **then**
4: $\quad x_{\text{pad}} \leftarrow x_1$
5: **else**
6: $\quad x_{\text{pad}} \leftarrow x_2$
7: $m \leftarrow$ random integer in $[2^{31}, 2^{32})$
8: $a = \lfloor x_{\text{pad}}/m \rfloor$, $b \leftarrow x_{\text{pad}} \bmod m$
9: $t_1 \leftarrow g^a$
10: $t_2 \leftarrow {t_1}^m$
11: $t_3 \leftarrow g^{b+2^{32}}$
12: **return** $t_2 \cdot t_3$ ▷ $g^{am} \cdot g^{b+2^{32}} = g^x$

B.2 SPAKE2+

The SPAKE2+ protocol [6] is another PAKE protocol similar to the SRP protocol and shares properties such as protection against an eavesdropper or a Man in the Middle. In Apple's library, it is solely used with elliptic curves.

The attack on SRP can be adapted to work with this protocol, and there are a few differences. The first is that two exponents are derived from the password, and, according to the source code of the library, the client computes two distinct exponentiations with these values. Since the exponentiation is vulnerable, it gives two distinguishers to run an offline dictionary attack. In the situation of the first variant, where the attacker is only an observer, this makes the filtering more effective.

References

1. Aldaya, A.C., García, C.P., Brumley, B.B.: From A to Z: projective coordinates leakage in the wild. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(3), 428–453 (2020). https://doi.org/10.13154/tches.v2020.i3.428-453
2. Aranha, D.F., Novaes, F.R., Takahashi, A., Tibouchi, M., Yarom, Y.: Ladderleak: Breaking ECDSA with less than one bit of nonce leakage. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) CCS '20: 2020 ACM SIGSAC. pp. 225–242. ACM (2020). https://doi.org/10.1145/3372297.3417268
3. Boudot, F., Gaudry, P., Guillevic, A., Heninger, N., Thomé, E., Zimmermann, P.: Comparing the difficulty of factorization and discrete logarithm: A 240-digit experiment. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12171, pp. 62–91. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56880-1_3
4. Braga, D.D.A., Fouque, P., Sabt, M.: Dragonblood is still leaking: Practical cache-based side-channel in the wild. In: ACSAC '20. pp. 291–303. ACM (2020). https://doi.org/10.1145/3427228.3427295

5. Brumley, B.B., Tuveri, N.: Remote timing attacks are still practical. In: Atluri, V., Díaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 355–371. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23822-2_20
6. Cash, D., Kiltz, E., Shoup, V.: The twin Diffie-Hellman problem and applications. J. Cryptol. **22**(4), 470–504 (2009). https://doi.org/10.1007/s00145-009-9041-6
7. Ciet, M., Joye, M.: (Virtually) free randomization techniques for elliptic curve cryptography. In: Qing, S., Gollmann, D., Zhou, J. (eds.) ICICS 2003. LNCS, vol. 2836, pp. 348–359. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39927-8_32
8. Cohen, H., Frey, G., Avanzi, R., Doche, C., Lange, T., Nguyen, K., Vercauteren, F. (eds.): Handbook of Elliptic and Hyperelliptic Curve Cryptography. Chapman and Hall/CRC (2005). https://doi.org/10.1201/9781420034981
9. Coron, J.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Koç, Ç.K., Paar, C. (eds.) CHES'99. LNCS, vol. 1717, pp. 292–302. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48059-5_25
10. Feix, B., Roussellet, M., Venelli, A.: Side-channel analysis on blinded regular scalar multiplications. In: Meier, W., Mukhopadhyay, D. (eds.) INDOCRYPT 2014. LNCS, vol. 8885, pp. 3–20. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13039-2_1
11. Genkin, D., Pachmanov, L., Pipman, I., Tromer, E., Yarom, Y.: ECDSA key extraction from mobile devices via nonintrusive physical side channels. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM SIGSAC 2016. pp. 1626–1638. ACM (2016). https://doi.org/10.1145/2976749.2978353
12. Goudarzi, D., Rivain, M., Vergnaud, D.: Lattice attacks against elliptic-curve signatures with blinded scalar multiplication. In: Avanzi, R., Heys, H.M. (eds.) SAC 2016. LNCS, vol. 10532, pp. 120–139. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-69453-5_7
13. Goundar, R.R., Joye, M., Miyaji, A., Rivain, M., Venelli, A.: Scalar multiplication on weierstraß elliptic curves from co-Z arithmetic. J. Cryptogr. Eng. **1**(2), 161–176 (2011). https://doi.org/10.1007/s13389-011-0012-0
14. Jancar, J., Sedlacek, V., Svenda, P., Sýs, M.: Minerva: The curse of ECDSA nonces systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(4), 281–308 (2020). https://doi.org/10.13154/tches.v2020.i4.281-308
15. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO '96. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
16. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) CRYPTO '99. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
17. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation **48**, 243–264 (1987)
18. Muller, F., Valette, F.: High-order attacks against the exponent splitting protection. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 315–329. Springer, Heidelberg (2006). https://doi.org/10.1007/11745853_21
19. Proton Technologies AG: Protonmail v3.6 release notes
20. Roche, T., Imbert, L., Lomné, V.: Side-channel attacks on blinded scalar multiplications revisited. In: Belaïd, S., Güneysu, T. (eds.) CARDIS 2019. LNCS, vol. 11833, pp. 95–108. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-42068-0_6

21. Taylor, D., Wu, T., Mavrogiannopoulos, N., Perrin, T.: Using the Secure Remote Password (SRP) protocol for TLS authentication. RFC **5054**, 1–24 (2007). https://doi.org/10.17487/RFC5054
22. Vanhoef, M., Ronen, E.: Dragonblood: Analyzing the Dragonfly handshake of WPA3 and EAP-pwd. In: 2020 IEEE Symposium on Security and Privacy, SP 2020. pp. 517–533. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00031
23. Wilson, E.B.: Probable inference, the law of succession, and statistical inference. Journal of the American Statistical Association **22**(158), 209–212 (1927). https://doi.org/10.1080/01621459.1927.10502953
24. Wu, T.: The SRP authentication and key exchange system. RFC **2945**, 1–8 (2000). https://doi.org/10.17487/RFC2945
25. Wu, T.D.: The Secure Remote Password protocol. In: NDSS 1998. The Internet Society (1998), https://www.ndss-symposium.org/ndss1998/secure-remote-password-protocol/