On Security of Key Derivation Functions in Password-based Cryptography

Gaurav Kodwani
Department of Computer Science
University at Albany, SUNY
Albany, NY, USA
gaurav.j.kodwani@gmail.com

Shashank Arora
Department of Computer Science
University at Albany, SUNY
Albany, NY, USA
sarora3@albany.edu

Pradeep K. Atrey
Department of Computer Science
University at Albany, SUNY
Albany, NY, USA
patrey@albany.edu

***Abstract*—Most common user authentication methods use some form of password or a combination of passwords. However, encryption schemes are generally not directly compatible with user passwords and thus, Password-Based Key Derivation Functions (PBKDFs) are used to convert user passwords into cryptographic keys. In this paper, we analyze the theoretical security of PBKDF2 and present two vulnerabilities, γ-*collision* and δ-*collision*. Using AES-128 as our exemplar, we show that due to γ-*collision*, text encrypted with one user password can be decrypted with $\gamma - 1$ different passwords. We also provide a proof that finding a collision in the derived key for AES-128 requires δ lesser calls to PBKDF2 than the known Birthday attack. Due to this, it is possible to break password-based AES-128 in $O(2^{64})$ calls, which is equivalent to brute-forcing DES.**

***Index Terms*—PBKDF, Password-based Cryptography, Security**

I. Introduction

Commonly used and popular symmetric key encryption scheme, AES-128 requires a random key of 128 bits. There are many AES-based applications in which users typically provide a password rather than a 128 bit random key. Due to low entropy and poor randomness, such passwords can not be considered adequately secure [3] [5]. Moreover, passwords are generally selected from a relatively small character set that can lead to a dictionary attack [5] [15] [6]. To avoid this, family of Password-based Key Derivation Functions (PBKDFs) was introduced in RFC 8018 [14].

PBKDFs make use of user provided passwords with a randomly generated salt and iteration count to derive cryptographic keys for use in encryption systems. The goal of these key derivation functions is to mitigate the need for managing large cryptographic keys. This helps users in better key management. The specifications of key derivation functions, PBKDF1 and PBKDF2 are listed in RFC 8018 [14]. According to the specifications, both functions require a pseudo-random function (PRF) to generate a pseudo-random, or seemingly random, bit-string from a seed. The user provided password acts as a seed to the PRF. RFC 8018 has various recommendations of popular cryptographic functions to use as PRFs. One of the recommendations is Hash based Message Authentication Code (HMAC) with variation of the Secure Hash Algorithm (SHA) for generating pseudo-random bit string [14].

978-1-7281-5684-2/20/$31.00 ©2021 IEEE

Implementation specifications given in RFC 8018 say "password-based key derivation is a function of password, salt and iteration count, where the latter two parameters need not be kept secret" [14]. Salt is a randomly generated value with the motivation of increasing the search space to avoid a dictionary attack. Iteration count specifies the number of times the underlying PRF needs to be executed in order to generate the key. Intention for this is to add CPU-intensive operations, so that it is infeasible for an attacker to do an exhaustive search.

NIST [5] and RFC 8018 [14] recommend the use of PBKDF2 in new application development. Also, they recommended the use of minimum 128-bit randomly-generated salt and minimum iteration count of 1000. Many libraries have been developed in different programming languages based on these recommendations and the implementation details specified in these documents. As a result, the PBKDF2 is widely used in different systems such as Wi-Fi Protected Access (WPA and WPA2) [7], Apple's iOS mobile operating system [2], and Firefox Sync for client-side password stretching [1]. Unfortunately, the specified way of deriving a cryptographic key from PBKDF2 using hashing algorithms as PRF has security vulnerabilities.

In this paper, we present two different but inter-related vulnerabilities in the family of PBKDFs that reduce the security of key generated for password based AES-128 equivalent to brute-forcing Data Encryption Standard (DES). The first of the discovered vulnerabilities has its roots in the Birthday attack [12], which has been traditionally used to find collisions in cryptographic hash functions. We have furthered this analysis and used the principles of Birthday attack to find security weakness in password-based encryption irrespective of bit-size of password or salt. According to our findings, it is possible to find multiple user provided passwords, such that text encrypted using one can be decrypted using the another password. Moreover, there exist entire set of user provided passwords that can be used interchangeably to encrypt and decrypt the same text. The exact number depends on the length of cryptographic key being derived and the hash function being used as the PRF. We call this vulnerability as γ-*collision*,

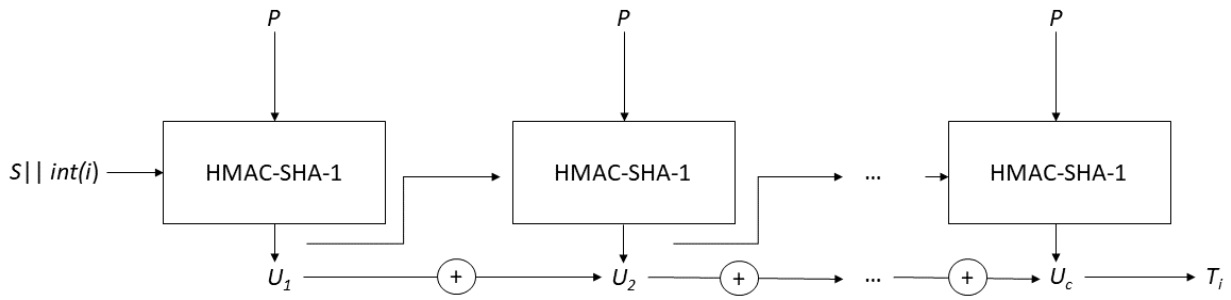

Fig. 1: Working of PBKDF2

where γ represents the number of different passwords that can be used to decrypt a text which is encrypted using a given user password. For instance, for a 128-bit cryptographic key derived using HMAC-SHA-1 that yields 160-bit output, the value of γ is 2^{32}. The size of such set of passwords is greater if HMAC-SHA-256 or HMAC-SHA-512 is used instead.

The second vulnerability, called δ-*cutback*, relates to a pair of passwords that lead to the same cryptographic key. We posit that such pair of passwords can be found by making reduced number of calls to PBKDF2. Here, δ represents the reduction in the number of calls. Note that the more the number of calls, the more the computational security it possesses. Our analysis of generating key for AES-128 suggests that finding such a pair is computationally as feasible as brute-forcing DES algorithm, therefore questioning the security of AES when used with PBKDF2.

The rest of the paper is organized as follows. In Section II, we not only present the background of PBKDF2 and Birthday attack, but also discuss the existing work that have analyzed the security of PBKDFs when associated with AES. Next in Section III, we present the two identified vulnerabilities with the detailed analysis and proofs. Further in Section IV, implications of these vulnerabilities are discussed. Finally, the paper is concluded in Section V.

II. Background and Related Work

A. Background

1) PBKDF2: PBKDF2 is a key-stretching algorithm [14] that converts password *P* (a user provided password or passphrase) into derived key $\mathcal{K}$ (a cryptographic key derived from password *P*). It takes in four input arguments: password *P*, salt *S*, iteration count *C*, and *dkLen* (length of $\mathcal{K}$), and outputs derived key $\mathcal{K}$ by repeatedly applying a pseudo-random function (PRF) for *C* times. From the input arguments, only *P* needs to be kept secret by an authentic user [14].

PBKDF2 supports the use of HMAC with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256 as the PRF [14]. Although, there is no limitation in the length of the derived key $\mathcal{K}$ in PBKDF2, it is limited by the hash function used as PRF. The length of derived key *dkLen*, must be less than $(2^{32}-1)\times$ *hLen*, where *hLen* is the output length of the PRF.

In this paper, we present our analysis with HMAC-SHA-1 as the choice of PRF. A separate analysis is provided later in the paper if HMAC-SHA-256 is used instead. The steps followed for key derivation in PBKDF2 are as follows.

Let l be the number of $hLen$ sized blocks required to derive $\mathcal{K}$ and r be the number of bits required from the last block. We calculate l and r by using the following two equations.

$$l = \lceil dkLen/hLen \rceil \tag{1}$$

$$r = dkLen - (l-1) \times hLen \tag{2}$$

Each block of the derived key is computed by function F using password P, salt S, iteration count C, and the index of the blocks i.

$$\begin{aligned} T_1 &= F(P,S,C,1) \\ T_2 &= F(P,S,C,2) \\ &\cdots \\ T_l &= F(P,S,C,l) \end{aligned}$$

As depicted in Figure 1, function F is an exclusive-or sum of C iterations of the PRF. Inputs for the PRF are: password P and the concatenation of salt S and the index value of the block i.

$$F(P,S,C,i) = (U_1 \oplus U_2 \oplus .. \oplus U_C) \tag{3}$$

where,

$$\begin{aligned} U_1 &= PRF(P, S||int(i)) && 1^{st} \text{ iteration} \\ U_2 &= PRF(P, U_1) && 2^{nd} \text{ iteration} \\ &\cdots \\ U_C &= PRF(P, U_{C-1}) && C^{th} \text{ iteration} \end{aligned}$$

Here, $int(i)$ stands for 32-bit integer value of i. Finally, the derived key is computed by concatenating T_1 though T_l. From the last block T_l, first *r* bits are picked, depending on the length of derived key, *dkLen*.

$$\mathcal{K} = T_1||T_2||...||T_l < 0...(r-1) > \tag{4}$$

TABLE I: Number of calls required to find a collision in PBKDF2 with different probability values

Probability	N
0.50	$1.177 \times 2^{n/2}$
0.75	$1.665 \times 2^{n/2}$
0.90	$2.145 \times 2^{n/2}$
0.95	$2.448 \times 2^{n/2}$
0.9999	$4.292 \times 2^{n/2}$

2) Birthday Attack: Birthday attack is a cryptanalytic technique to find collisions in hash functions using birthday paradox in probability theory [12]. This attack helps us to calculate the number of computations required to find two input values that lead to generation of the same output (known as Collision) of a hash function, with a certain probability. A hash function f maps inputs of arbitrary length with fix sized output of n bits. The function f can generate H unique values where $H = 2^n$.

Birthday attack says that *"It is possible to find a collision in* $1.18 \times \sqrt{2^n}$ *computations of a hash function, with* 2^n *being the classical pre-image resistance security with approximately 50% probability"*. It is important to note that in this attack, only the final output value of the algorithm is observed. Now, by the definition of Birthday attack, the number of hash values that need to be computed to encounter a collision with a probability of p is given as follows:

$$N(p; H) \approx \sqrt{2H \times ln\frac{1}{1-p}}$$

Now, if the probability $p = 0.5$ and $H = 2^n$, minimum number of hash values need to be calculated are as follows:

$$\begin{aligned} N(0.5; 2^n) &\approx \sqrt{2 \times 2^n \times ln\frac{1}{1-0.5}} \\ &\approx \sqrt{2 \times 2^n \times 0.693} \\ &\approx \sqrt{2^n \times 1.386} \\ &\approx 1.177\sqrt{2^n} \\ &\approx 1.77 \times 2^{n/2} \end{aligned}$$

The value *N* is known as *Birthday Bound* for n-bit generating hash function f. Table I shows values of *N* for different *p* values.

B. Related Work

Authors in [17], [6] [14] discussed the importance of and provided recommendations for length of salt (S) and iteration count (C). We, however, show later in this paper that increasing the length of S or C does not subvert the discovered vulnerability of PBKDFs. In [13], the authors proposed the use of a larger character set for user passwords to increase the security of PBKDFs. On the contrary, our analysis proves that the number of passwords that can decrypt a ciphertext and the number of calls to PBKDFs to find such a password is independent of the input character set. In [9] and [18] Zhou et al. suggested the use of user passwords longer than 6 characters to make PBKDFs practically safe to use. According to our analysis, the length of user password does not impact the vulnerability in PBKDFs that use HMAC as PRF. In [10], the authors proposed a modified PBKDF scheme. While the scheme works better against GPU accelerated attacks, it however suffers from the same vulnerabilities presented in this paper.

III. Vulnerabilities in PBKDFs

In this section, we discuss the two vulnerabilities, γ-$collision$ and δ-$cutback$, which we discovered in the PBKDF2. Our analysis is based on the following assumptions:

1) There is no structural problem in the Hash function (SHA) used in PBKDFs i.e. the best possible cryptanalytic attack on SHA family is Birthday attack.
2) The value of iteration count C is known to the attacker. This value is constant and specified in the implementation [13].

A. Vulnerability 1: γ-collision

When using password based cryptography with PBKDFs, there exist γ passwords that lead to the same derived cryptographic key $\mathcal{K}$. This vulnerability enables an attacker to find multiple potential user passwords, such that text encrypted using one user password can be decrypted using another password. Additionally, it is not just two, but the entire sets of user passwords exist that can be used interchangeably to encrypt and decrypt the same text.

The value of γ depends on the difference of the output length of PRF and the number of bits in the last block that are included in the derived key $\mathcal{K}$ i.e. $\gamma = 2^{hLen-r}$. For a 128-bit cryptographic key (like in AES-128), deriving the key using HMAC-SHA-1 as the PRF, the value of γ is 2^{32}. The proof is as follows.

Theorem 1. *For a cryptographic key of* $dkLen$ *size, deriving the key using the PRF of* $hLen$ *hash output, the value of* γ *is* 2^{hLen-r}.

Proof. We begin by calculating the l and r using equations (1) and (2).

Let password P_1, salt S_1 and iteration count C_1 be the inputs to PBKDF2 to derive the key $\mathcal{K}_1$ of $dkLen$ bits. Using equation (4), $\mathcal{K}_1$ is calculated as follows:

$$\mathcal{K}_1 = T_1^1||T_2^1||...||T_l^1 < 0...(r-1) >$$

Now in the final step, the concatenation of all T^1 blocks with the first r bits of T_l^1 form the derived key $\mathcal{K}_1$ and the last $hLen-r$ bits from the T_l^1th block are truncated. The truncated string $Trunc_1$ from the $\mathcal{K}_1$ is determined as follows.

$$Trunc_1 = T_l^1 < r...(hLen-1) >$$

Similarly, we can calculate $\mathcal{K}_2$ and $Trunc_2$ by passing password P_2 ($P_1 \neq P_2$), salt S_2 ($S_1 \neq S_2$) and iteration count C_1 as input arguments to PBKDF2.

$$\mathcal{K}_2 = T_1^2||T_2^2||...||T_l^2 < 0...(r-1) >$$

$$Trunc_2 = T_l^2 < r..(hLen-1) >$$

Let us assume that $\mathcal{K}_1 = \mathcal{K}_2$ and $Trunc_1 \neq Trunc_2$. Since PBKDF2 uses HMAC-SHAs as the PRF and the input passwords and salts are different, it results in $T_l^1 \neq T_l^2$.

Now, since T_1^1 to T_l^1 are generated by iterating and XORing the PRF outputs C_1 times, the total number of unique input and output pairs are 2^{hLen}. However, the key is derived by selecting first $dkLen$ bits out of $hLen$ bits of the concatenated string $T_1||T_2||...||T_l$, which means every time the last r bits are truncated when deriving the key.

Therefore, 2^{hLen-r} different values of T_l^1 derive the same key. Alternatively, we can say 2^{hLen-r} different passwords lead to the same derived key. □

Example: To derive a key of 128 bits for AES-128 using PBKDF2 and PRF as HMAC-SHA-1. The output size of HMAC-SHA-1 is 160 bits.

Here, the size of the derived key is less than the output block size. Therefore, the value of l is 1. We have to select r bits from the last block, which can be calculated as follows:

$$l = \lceil dkLen/hLen \rceil = \lceil 128/160 \rceil = 1$$
$$r = (dkLen - (l-1) \times hLen)$$
$$r = (128 - (1-1) \times 160) = 128$$

Let password P_1, salt S_1 and iteration count C_1 be the inputs to PBKDF2 to derive the key $\mathcal{K}_1$ of 128 bits. Calculations are as follows:

$$U_1^1 = HMAC - SHA - 1(P_1, S_1 \,||\, int(1)) \quad 160 \text{ bits}$$
$$U_2^1 = HMAC - SHA - 1(P_1, U_1) \quad 160 \text{ bits}$$
$$\cdots$$
$$U_{C_1}^1 = HMAC - SHA - 1(P_1, U_{C_1-1}) \quad 160 \text{ bits}$$

$$F^1(P_1, S_1, C_1, 1) = (U_1 \oplus U_2 \oplus .. \oplus U_{C_1}) \quad 160 \text{ bits}$$

$$T_1^1 = F^1(P_1, S_1, C_1, 1) \quad 160 \text{ bits}$$

Now in the final step, first r bits of T_1^1 form the derived key $\mathcal{K}_1$ and the last $hLen - r$ bits (32 bits for AES-128) are truncated.

$$\mathcal{K}_1 = T_1^1 < 0..(r-1) >$$
$$\mathcal{K}_1 = T_1^1 < 0..127 >$$
$$Trunc_1 = T_1^1 < r..(hLen-1) >$$
$$Trunc_1 = T_1^1 < 128..159 >$$

Similarly,

$$\mathcal{K}_2 = T_1^2 < 0..127 >$$
$$Trunc_2 = T_1^2 < 128..159 >$$

where,

$$T_1^2 = F(P_2, S_2, C_1, 1)$$

Now, an assumption that $\mathcal{K}_1 = \mathcal{K}_2$ and $Trunc_1 \neq Trunc_2$, leads to $T_1^1 \neq T_1^2$.

In this case, the total number of unique input and output pairs are 2^{160}. However, the key is derived by selecting first 128 bits out of 160 bits of T_1^1, which means every time the last 32 bits are irrelevant when deriving the key. Therefore, each derived key would have 2^{32} different values of T_1^1.

TABLE II: The values of γ and δ for different *dkLen* values

dkLen	γ	δ
128	2^{32}	2^{16}
192	2^{128}	2^{64}
256	2^{64}	2^{32}

Table II shows the γ values for different *dkLen*. For $dkLen = 128$, $\gamma = 2^{32}$; for $dkLen = 192$, $\gamma = 2^{128}$ and for $dkLen = 256$, $\gamma = 2^{64}$, computed using Theorem 1. It can be observed that generating keys for AES-192 and AES-256 using PBKDF2 with HMAC-SHA-1 results in an even higher γ value as compared to AES-128.

B. Vulnerability 2: δ-cutback

A pair of passwords that lead to the same cryptographic key can be found by making δ lesser calls to PBKDFs than the Birthday attack on PBKDFs. This reduction in number of calls is the δ-$cutback$. The vulnerability has its roots in the Birthday attack [12]. The Birthday attack has been traditionally used to find collisions in cryptographic hash functions. We use the principles of this attack to find the vulnerability in password based cryptography. The proof is given as follows.

Theorem 2. *A pair of passwords that lead to the same cryptographic key can be found in $O(2^{(l \times hLen - dkLen)/2})$ lesser calls to PBKDF2 than the Birthday attack on PBKDF2, thus resulting in $O(2^{(l \times hLen - dkLen)/2})$-cutback.*

Proof. The class of PBKDFs are composed of a hash function and bit manipulation for several iterations to produce a fixed length output. Thus, PBKDFs can also be considered as a hash function in itself. Due to this, they are susceptible to the Birthday attack.

PBKDF2 can generate an output of maximum $l \times hLen$ bits, as shown in equation (4). Therefore, according to Birthday attack, $O(2^{(l \times hLen)/2})$ calls to PBKDFs would lead to a collision.

As shown in equation (4), the last r bits of block T_l are truncated and play no part in derived key $\mathcal{K}$. This reduces the effective length of the output of PBKDFs to $dkLen$ bits. As we consider PBKDFs to be hash function in itself. Therefore, we can say that the collision can be found in $O(2^{dkLen/2})$ calls to PBKDF2, according to the Birthday attack. Thus, δ can be determined as follows:

$$\delta = \frac{O(2^{(l \times hLen)/2})}{O(2^{dkLen/2})}$$
$$= O(2^{(l \times hLen)/2 - dkLen/2})$$
$$= O(2^{(l \times hLen - dkLen)/2})$$

Thus, PBKDF2 suffers from $O(2^{(l \times hLen - dkLen)/2}$-*cutback*. □

Example: Let us consider an example of deriving a key of 128 bits for AES-128 using PBKDF2 and PRF as HMAC-SHA-1. The output bits size of HMAC-SHA-1 is 160 bits.

First, we consider PBKDF2 as a hash function and apply Birthday attack on it. The size of the derived key is lesser than the output block size. Therefore, the value of l is 1. The output size of HMAC-SHA-1, $hLen$, is 160 bits and it can generate 2^{160} unique hash values. Without any further analysis, by the definition of Birthday attack, a collision can be found in 1.18×2^{80} calls of PBKDF2 with 50% probability (Table I). Now, using the results from Table I, we can find pair of user provided passwords with 0.9999 probability that lead to the same key in 4.292×2^{80} calls of PBKDF2. This requires lesser effort than the best known attack for AES-128, Biclique [8].

Second, according to Theorem 2, we know that when deriving key for AES-128 using PBKDF2 with HMAC-SHA-1, the algorithm can generate 2^{128} unique values. So, by using the results of Table I, to find a pair of user provided passwords that lead to the same 128-bit key with 0.9999 probability would require 4.292×2^{64} calls to PBKDF2 as opposed to 4.292×2^{80}. This leads to a $O(2^{(160-128)/2}) = O(2^{16})$-*cutback*. That means collision can be found in 2^{16} lesser calls than the Birthday attack.

As shown in Table II, the value of δ is higher for *dkLen* lower and higher than 128 bits when using PBKDF2 with HMAC-SHA-1. Additionally, an increase in *hLen* results in a significant increase in δ, as shown in Table III. For $dkLen = 64$, $\delta = 2^{48}$; for $dkLen = 192$, $\delta = 2^{64}$ and for $dkLen = 256$, $\delta = 2^{32}$, resulting in an even higher cutback. Moreover, for AES-128, when using PBKDF2 with HMAC-SHA-1, we require $\mathcal{O}(2^{64})$ calls to PBKDFs to find a collision, which is computationally equivalent to brute-forcing the DES algorithm.

IV. Implications and Discussion

Due to the aforementioned vulnerabilities in PBKDFs, it is safe to assume that they are not as secure as previously considered. In this section, we explore the implications on γ-$collision$ and δ-$cutback$ to potential solutions for increasing the security of PBKDFs. We explore two such potential solutions and argue that they are inadequate.

A. Using Alternative Hash Functions in PBKDFs

After looking at the vulnerabilities for PBKDFs discussed in Section III, one of the most trivial solutions could be to use more secure hash function like HMAC-SHA-256 or HMAC-SHA-512 as the PRF. The expectation by this change would be the probability of finding a collision will decrease as the hash function output size increases. However, since the output size of PBKDFs will remain the same, it will provide the same level of security as HMAC-SHA-1. We discuss it with an example below.

Example. Let us analyze the security of PBKDF2 using HMAC-SHA-1 and HMAC-SHA-512 as the PRF to derive the key for AES-128. The output size of HMAC-SHA-1 and HMAC-SHA-512 is 160 bits and 512 bits, respectively. Using the values of *hLen* from the results in Theorem 2, security of PBKDF2 using HMAC-SHA-1 can be determined by reducing δ calls from the calls required for Birthday attack, i.e. $2^{160/2}/2^{(1\times160-128)/2} = 2^{64}$. Similarly, for the security of HMAC-SHA-512, this number would be $2^{512/2}/2^{(1\times512-128)/2} = 2^{64}$, which is same as HMAC-SHA-1. Thus, using HMAC-SHA-256 and HMAC-SHA-512 increases the number of hash values that can lead to the same derived key $\mathcal{K}$.

TABLE III: The values of γ and δ for different *hLen* values

hLen	γ	δ
160	2^{32}	2^{16}
256	2^{128}	2^{64}
512	2^{384}	2^{192}

Table III shows the relationship between *hLen* and γ. Earlier, we showed that for HMAC-SHA-1 the output size of T_1 was 160. Also, the number of T_1 values that derives the same key is given by 2^{hLen-r}. So, in case of HMAC-SHA-256 and HMAC-SHA-512, used with AES-128 will have $\gamma = 2^{256-128}$ = 2^{128} and $2^{512-128}$ = 2^{384} different values of T_1 resulting in the same cryptographic key. The reason behind this is the many-to-one mapping between the hash values and the derived key. Therefore, by replacing the hash function with a greater output length hash function for the PRF ends up being more vulnerable. To find collision in any hash function, the number of function calls required depends upon the message digest (output size), not on the internal process or operations.

However, effort required to find a collision remains the same i.e. 4.292×2^{64}. This is because the length of final output ($dkLen$) is same as before and the effort required to find a collision is solely dependant upon the size of the final output of PBKDFs. Thus, as it stands the use of any hashing function as PRF in PBKDFs results in the above vulnerabilities.

B. Increasing Iteration Count

By now, it is clear that there is no way to increase the efforts required to find key pairs by using different hash functions as PRF. In [16], the authors corroborate our analysis that cascading hash functions does not lead to an increase in security. Mathematically, an iteration count of C increases the security strength of a password by $log_2(C)$ bits against trial-based attacks like brute-force or dictionary attacks [14] by increasing the execution time of PBKDFs. So, another solution that presents itself is increasing the number of times the PRF is called, i.e. increasing C.

Now the question is that is it be possible to make iteration count C large enough to make it infeasible for attacker to brute-force 2^{64} input values? Yes, it seems so. We can set the value of iteration count C to a large enough number to avoid brute-forcing the input values to find collision.

Let us assume that new iteration count is $C^{'}$. Theorem 3 shows the relationship between the strength of the derived key with iteration count C and new iteration count $C^{'}$.

Theorem 3. *Relationship between the new iteration count $C^{'}$ and the old iteration count C is $C^{'} = \delta \times C$.*

Proof. As mentioned in Theorem 2, applying Birthday attack on PBKDF2, once with and then without taking δ-*cutback* into account, reduces the number of iterations required to find a pair of passwords that lead to the same derived key $\mathcal{K}$ in the former case.

Also, as mentioned above, iteration count increases the security strength of a password by $log_2(C)$ bits. We thus establish the following relationship between C and $C^{'}$: $2^{log_2(C^{'})}\times$ Birthday attack with δ-*cutback* = $2^{log_2(C)}\times$ Birthday attack without δ-*cutback*. We argue it as follows:

$$2^{log_2(C^{'})} \times 4.292 \times 2^{dkLen/2} = 2^{log_2(C)} \times 4.292 \times 2^{(l \times hLen)/2}$$

$$2^{log_2(C^{'})} \times 2^{dkLen/2} = \frac{2^{log_2(C)} \times 4.292 \times 2^{(l \times hLen)/2}}{4.292}$$

$$2^{log_2(C^{'})} \times 2^{dkLen/2} = 2^{log_2(C)} \times 2^{(l \times hLen)/2}$$

$$log_2(C^{'}) + \frac{dkLen}{2} = log_2(C) + \frac{l \times hLen}{2}$$

$$log_2(C^{'}) - log_2(C) = \frac{l \times hLen}{2} - \frac{dkLen}{2}$$

$$log_2 \frac{C^{'}}{C} = \frac{l \times hLen - dkLen}{2}$$

$$\frac{C^{'}}{C} = 2^{(l \times hLen - dkLen)/2}$$

$$C^{'} = 2^{(l \times hLen - dkLen)/2} \times C$$

$$C^{'} = \delta \times C$$

□

The minimum value of iteration count, as recommended by RFC 8018 [14] and NIST [5] is 1000. Using the results from Theorem 3, new recommendation for minimum iteration count should be $1000 \times \delta$. In case of password based AES-128 when used with PBKDF2 and HMAC-SHA-1, $\delta = 2^{16}$, making the new recommended iteration count $1000 \times \delta = 1000 \times 2^{16}$ ($\approx 2^{26}$). This new iteration count will provide the required security, but it will be infeasible for many practical applications because this will greatly increase the execution time of PBKDFs.

Furthermore, increasing the iteration count is a temporary solution. In [4], the authors were able to evaluate $250,000$ passwords per second using multiple FPGAs with an iteration count of 2000, and in [11], performance of $356,352$ passwords per second was achieved for 1000 iterations. With GPUs and FPGAs becoming increasingly powerful, the execution time per iteration of PBKDFs is cut down significantly. Moreover, with the advent of Infrastructure as a Service (IaaS), it is becoming easier for an adversary to procure hardware powerful enough to counteract the execution time increase induced by increasing *C*. Thus, it is only a matter of time before the iteration count becomes insignificant.

V. Conclusion

Due to the two vulnerabilities γ-$collision$ and δ-$cutback$ in PBKDFs presented in this paper, the effective security of password-based AES-128 is compromised, and is equivalent to brute-forcing DES. The vulnerabilities exist due to the use of hashing functions as PRF and the way the cryptographic keys are derived from the output of the PRF. As it stands now, a so-called computationally secure encryption scheme, such as AES, may not have any known feasible attack but if used with key derivation functions like PBKDFs are prone to attack.

References

[1] Firefox sync's new security model. URL: https://blog.mozilla.org/services/2014/04/30/firefox-syncs-new-security-model/, accessed on 2021-05-15.

[2] iOS Security: iOS 12.3, May 2019. URL: https://www.apple.com.cn/business/docs/site/iOS_Security_Guide.pdf, accessed on 2021-05-15.

[3] Martín Abadi and Bogdan Warinschi. Password-based encryption analyzed. In *International Colloquium on Automata, Languages, and Programming*, pages 664–676, Lisbon, Portugal, 2005. Springer.

[4] Ayman Abbas, Rian Voß, Lars Wienbrandt, and Manfred Schimmler. An efficient implementation of PBKDF2 with RIPEMD-160 on multiple FPGAs. In *International Conference on Parallel and Distributed Systems*, pages 454–461, Hsinchu, Taiwan, 2014. IEEE.

[5] Elaine Barker and Allen Roginsky. Recommendation for cryptographic key generation. *NIST Special Publication*, 800:133, 2012.

[6] Mihir Bellare, Thomas Ristenpart, and Stefano Tessaro. Multi-instance security and its application to password-based cryptography. In *Annual Cryptology Conference*, pages 312–329, Santa Barbara, CA, USA,, 2012. Springer.

[7] Florent Bersani and Hannes Tschofenig. The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method. URL: https://tools.ietf.org/html/rfc4764, accessed on 2021-05-15.

[8] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 344–371, Seoul, South Korea, 2011. Springer.

[9] Jie Chen, Jun Zhou, Kun Pan, Shuqiang Lin, Cuicui Zhao, and Xiaochao Li. The security of key derivation functions in WINRAR. *JCP*, 8(9):2262–2268, 2013.

[10] Xiurong Chen, Xiaochao Li, Yihui Chen, Pengtao Li, Jianli Xing, and Lin Li. A modified PBKDF2-based MAC scheme XKDF. In *IEEE Region 10 International Conference TENCON*, pages 1–6, 2015.

[11] Markus Dürmuth, Tim Güneysu, Markus Kasper, Christof Paar, Tolga Yalcin, and Ralf Zimmermann. Evaluation of standardized password-based key derivation against parallel processing platforms. In *European Symposium on Research in Computer Security*, pages 716–733, Pisa, Italy, 2012. Springer.

[12] Marc Girault, Robert Cohen, and Mireille Campana. A generalized birthday attack. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 129–156, Davos, Switzerland, 1988. Springer.

[13] Xiaochao Li, Cuicui Zhao, Kun Pan, Shuqiang Lin, Xiurong Chen, Benbin Chen, Deguang Le, and Donghui Guo. On the security analysis of PBKDF2 in OpenOffice. *Journal of Software*, 10(2):116–127, 2015.

[14] Kathleen Moriarty, Burt Kaliski, and Andreas Rusch. PKCS#5: Password-based cryptography specification version 2.1. Technical report, 2017.

[15] Robert Morris and Ken Thompson. Password security: A case history. *Communications of the ACM*, 22(11):594–597, 1979.

[16] Krzysztof Pietrzak. Non-trivial black-box combiners for collision-resistant hash-functions don't exist. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 23–33, Barcelona, Spain, 2007. Springer.

[17] Meltem Sönmez Turan, Elaine Barker, William Burr, and Lily Chen. Recommendation for password-based key derivation. *NIST special publication*, 800:132, 2010.

[18] Jun Zhou, Jie Chen, Kun Pan, Cuicui Zhao, and Xiaochao Li. On the security of key derivation functions in office. In *Anti-counterfeiting, Security, and Identification*, pages 1–5, Taipei, Taiwan, 2012. IEEE.