Design and Analysis of Password-Based Key Derivation Functions

Frances F. Yao[1] and Yiqun Lisa Yin[2]

[1] Department of Computer Science,
City University of Hong Kong,
Kowloon, Hong Kong
`csfyao@cityu.edu.hk`

[2] Princeton Architecture Laboratory for Multimedia and Security,
Princeton University,
Princeton, NJ 08544
`yyin@princeton.edu`

Abstract. A password-based key derivation function (KDF) – a function that derives cryptographic keys from a password – is necessary in many security applications. Like any password-based schemes, such KDFs are subject to key search attacks (often called dictionary attacks). Salt and iteration count are used in practice to significantly increase the workload of such attacks. These techniques have also been specified in widely adopted industry standards such as PKCS and IETF. Despite the importance and wide-spread usage, there has been no formal security analysis on existing constructions. In this paper, we propose a general security framework for password-based KDFs and introduce two security definitions each capturing a different attacking scenario. We study the most commonly used construction $H^{(c)}(p\|s)$ and prove that the iteration count c, when fixed, does have an effect of stretching the password p by $\log_2 c$ bits. We then analyze the two standardized KDFs in PKCS#5. We show that both are secure if the adversary cannot influence the parameters but subject to attacks otherwise. Finally, we propose a new password-based KDF that is provably secure even when the adversary has full control of the parameters.

1 Introduction

1.1 Background and Motivation

Cryptographic keys are essential in virtually all security application. In practice, however, inputs to an application are typically raw key materials, such as passwords, that are not yet in the form to be used as keys. Therefore, a key derivation function (KDF) – a function that derives cryptographic keys from keying materials – is often a necessary component in all security applications.

There are many usage scenarios for key derivation functions depending on the form of the input. For example, the input can be a user password, a random seed value from some entropy source, or an output value from a cryptographic operation such as Diffie-Hellman key agreement. The second scenario is typically handled by a pseudorandom number generator (e.g., the FIPS PRNG in [6]), and the third scenario is handled by hashing the long output down to the required key length (e.g., the KDF1 in [8]). In both scenarios, there is usually enough entropy in the key materials.

In this paper, we focus our study on password-based key derivation functions. Unlike the other two scenarios mentioned above, passwords, in particular those chosen by a user, often are short or have low entropy. Therefore, special treatment is required in key derivation to defend against exhaustive key search attacks.

One basic approach for designing a password-based key derivation function is to derive the key from the password p and a random known value s (called salt), by applying a function H (such as hash, keyed hash, or block cipher) for a number of iterations c (called iteration count). For example, the following is a typical construction[1]:

$$key = H^{(c)}(p\|s).$$

Intuitively, the salt s serves the purpose of creating a large set of possible keys corresponding to a given password, among which one key is selected according to the salt used in each execution of the KDF. The iteration count c serves the purpose of increasing the cost of deriving each key, thereby significantly increasing the workload of key search attacks. These two techniques have been commonly used in practice and also specified in widely adopted industry standards, including PKCS [13] and IETF [4].

In a more general setting, we can view a password-based KDF as a method for stretching any short keys (not necessarily passwords) into longer keys. An efficient key stretching method can be useful in strengthening security without any architectural or policy changes for complex systems (e.g., a credit-card system). In addition, password-based KDF can be used in a straightforward way to define password-based encryption schemes and message authentication schemes.

Despite their importance and wide-spread usage, there has been no formal security analysis of existing password-based KDFs. Furthermore, there has been no general security framework for analyzing such KDFs. It is possible that the above popular KDF construction is insecure against some sophisticated key-search attacks even though the parameters are chosen large enough and the underlying hash function H is sound (e.g., it can be considered as a random oracle).

1.2 Our Framework

In this paper, we propose a general security framework for studying password-based key derivation functions. Our framework aims at capturing various types of key-search attacks and allowing concrete analysis of attacker's success probability relating to its available computational resource for launching key-search attacks. We also model salt and iteration count in a way as they are used in practice so that their impact on the overall security of the scheme can be quantified.

[1] Throughout the paper, we use $H^{(c)}$ to denote that H is applied c times and use $\|$ to denote the concatenation of two strings.

Key search attacks on password-based KDFs can be quite sophisticated [10], but they generally target at the *construction* of the key derivation function F and treat the underlying function H as a black-box transformation. This motivates us to model the underlying primitive H as a random oracle [2] and hence its internal structure is ignored in the analysis.

We define two levels of security for the KDF depending on the capability of the adversary A: – in the *weak* model A can only observe the output of F while in the *strong* model A can query F on inputs of its choice. In both models, A can make queries to H, and the maximum number of such queries captures A's available computational power to launch key search attacks. Roughly, the construction of F is secure under each model if A with the given capability cannot distinguish the output of F for an unknown password p from a random string.

1.3 Main Results

Using the proposed framework, we first study the security of the iterative construction. We prove that if an adversary A makes at most t queries to H, then the success probability Adv that it can distinguish the derived key for an unknown p from a random string of the same length n satisfies

$$\frac{\lfloor t/c \rfloor}{|PW|} < Adv < \frac{\lfloor t/c \rfloor}{|PW|} + \frac{t^2}{2^n}.$$

The upper bound is dominated by the first term, since the second term is negligible in practical settings. The above result implies that, for a fixed iteration count, there is no short-cut in the key search other than computing H iteratively for each password. In other words, the iteration count c increases the workload of exhaustive key search by a factor of c. So the iteration construction effectively stretches a k-bit key to a $(k + \log_2 c)$-bit key.

We then focus our attention on practical password-based KDFs and analyze the two KDFs in PKCS#5 [13] – the *de facto* standard for password-based cryptography. Our analysis on the iterative construction implies that the two KDFs are weakly secure as long as the computational resource available to the attacker is much less than $c|PW|$ (although it can be much larger than $|PW|$). We also show that neither KDF is secure in the strong model and discuss how such security weakness may be exploited to mount attacks in practical scenarios.

Based on the insight gained from our earlier analysis, we propose a new password-based key derivation function with enhanced security. The main idea in our construction is to include iteration count explicitly in the input to the derivation function to prevent it from being manipulated by the attacker. We show that the new KDF is secure in the strong model.

1.4 Related Work

Rigorous analysis of password-based key derivation schemes seems to have received relatively little attention compared to other cryptographic schemes. In

[10], the term *key-stretching* is used for conversion of low-entropy keys into longer keys by mechanisms such as iterated hash. A connection was made between the cost of computing $H^{(c)}$ and the cost of finding a collision for H. Roughly speaking, if $H^{(c)}$ could be computed on average with fewer than $c/2$ calls to H, then this would lead to a collision search for H faster than the naive birthday attack. Although it would be hard to translate the result into a standard concrete security model, it is certainly of practical interest. Our results on $H^{(c)}$ are well quantified; moreover, the framework for KDF is rigorously defined and the effects of salt and iteration are studied in a standard *distinguish-from-random* model with respect to specific query types.

In [14], the UNIX password hashing algorithm is analyzed. The core of the algorithm is roughly $f(p) = DES_p^{(25)}(0)$, that is, to encrypt the value zero 25 times using the password p as the key. So the algorithm has an iterative structure somewhat similar to the password-based KDFs considered here. It is proved that the algorithm is a secure hashing function if DES is a secure block cipher. However, as pointed out by the authors, their analysis only implies that iteration does not harm security, but they are not able to show that iteration actually enhances security as one would intuitively expect.

Key derivation functions in a general sense share some similarity in their design, such as the use of hash functions to process the raw key materials. For instance, the pseudorandom number generator (PRNG) defined in FIPS [6] is hash-based and can derive long keys from a random seed. The exact construction, however, is quite different: in the password-based KDF the hash is applied iteratively while in the PRNG the hash outputs are concatenated to produce the key. Therefore, security analysis of PRNG type of key derivation [5] does not directly apply to password-based key derivation.

Another related subject is password-based authentication and key exchange protocols, and the goal of such protocols is to authenticate two parties who share a common password. Existing protocols use various public-key techniques such as RSA or Diffie-Hellman in a w ay that the messages exchanged between the parties provide little or no help for an attacker to guess the password. A survey of well-analyzed protocols can be found in [9].

2 Security Framework and Definitions

2.1 KDF Model

We denote a password-based key derivation function as

$$y = F(p, s, c)$$

where p is password, s is salt, c is iteration count, and y is derived key of length n. Let the set of passwords, salts, and iteration counts be PW, S, and C. For a fixed $p \in PW$, we can view $y = F_p(s, c)$ as a function with input (s, c) and output y. We interchangeably write $F(p, s, c)$ or $F_p(s, c)$ depending on the context.

For ease of analysis, we make some assumptions on the sets PW, S, and C. We assume that $PW = \{0,1\}^l$, $S = \{0,1\}^s$, and $C = [c_*, c^*]$ for some integers $0 < c_* < c^*$. The upper limit ensures that the iteration count is not too large, since otherwise the KDF becomes too slow and useless. In addition, we assume that the length of $p\|s\|c$ is at most n, although our analysis can be extended to the more general case (see Section 4.3 for further discussions).

We denote the underlying primitive that is used to construct F as H. For example, H can be instantiated using practical hash functions as building blocks. Since key search attacks typically treats H as a black-box transformation without exploiting its internal structure, we model H as a random function from $\mathcal{R}_n$, the set of all functions from $\{0,1\}^n$ to $\{0,1\}^n$. That is, we focus our analysis on how F is constructed based on H rather than the structure of H itself.

2.2 Attack Model

Consider a typical usage scenario of a password-based KDF in which two users Alice and Bob share a password p. To encrypt a message, Alice sets s, c and derives a key $y = F(p, s, c)$. She then uses y to encrypt and obtain the ciphertext z. Alice sends (z, s, c) in the clear to Bob. Bob derives the key y by computing $y = F(p, s, c)$ and uses y to decrypt z.

From the attacker's point of view, the salt s and the count c are both *known*. The attacker usually does not have control of s or c, but in certain scenarios they can be *chosen*. The derived key y may be hidden from the attacker, but it can become *known* for various reasons (e.g., it was leaked out due to system security holes), in which case the attacker obtains a tuple (y, s, c) corresponding to some unknown password p.

In our attack model, we assume that s and c can be either known or chosen, and the derived key y is always known. An attacker A to a password-based KDF is a polynomial-time algorithm that may use the following two types of oracle queries:

- *H Query*: Query the underlying function H on input x and obtain $H(x)$. That is, A has access to oracle $H(.)$.
- *F Query*: For an unknown password p, query the key derivation function F on input (s, c) and obtain the derived key $y = F_p(s, c)$. That is, A has access to oracle $F_p(.,.)$ for some unknown p.

In practice, the number of H queries can be quite large, since it is determined by the adversary's available computational resource for performing an offline key search attack. In contrast, the number of F queries is very limited, since it is usually determined by the security design and policy of the system, not the adversary.

2.3 Security Definition

We introduce two security definitions – weakly secure and strongly secure – depending on attacker's capability. In the weak model, we assume that the attacker

A can make only queries to H, while in the strong model, we assume that A can make queries to both H and F. The goal of the attacker A is to distinguish the derived key $y = F_p(s, c)$ for $p \overset{r}{\leftarrow} PW$ from a random string of the same length[2].

Definition 1. Weakly Secure KDF. *Let $y = F_p(s, c)$ be a password-based KDF. Let $b \in \{0, 1\}$. We consider the following experiment depending on b:*

```
Experiment $\mathbf{E}_b$
    $p_0 \overset{r}{\leftarrow} PW$ // password is generated at random
    $H \overset{r}{\leftarrow} \mathcal{R}_n$ // $H$ is generated at random
    $s_0 \leftarrow S, c_0 \leftarrow C$ // salt and count are fixed and known
    If $b = 0$, then $y_0 \leftarrow F_{p_0}(s_0, c_0)$, else $y_0 \overset{r}{\leftarrow} \{0,1\}^n$
    $i \leftarrow 0$
    repeat
        $i \leftarrow i + 1$
        $A$ chooses $x_i$ and is given $H(x_i)$
    until $A$ reaches the maximum number of queries
    $A$ outputs either 0 or 1
```

The success probability of A is defined as

$$Adv_A(t) = Pr_{E_1}[A = 1] - Pr_{E_0}[A = 1].$$

where t denote the maximum number of queries to H. The maximum success probability achievable by any adversary A is denoted by $Adv(t)$.

Definition 2. Strongly Secure KDF. *Let $y = F_p(s, c)$ be a password-based KDF. Let $b \in \{0, 1\}$. We consider the following experiment depending on b:*

```
Experiment $\mathbf{E}_b$
    $p_0 \overset{r}{\leftarrow} PW$ // password is generated at random
    $H \overset{r}{\leftarrow} \mathcal{R}_n$ // $H$ is generated at random
    $s_0 \leftarrow S, c_0 \leftarrow C$ // salt and count are fixed and known
    If $b = 0$, then $y_0 \leftarrow F_{p_0}(s_0, c_0)$, else $y_0 \overset{r}{\leftarrow} \{0,1\}^n$
    $i \leftarrow 0$
    repeat
        $i \leftarrow i + 1$
        $A$ first decides which type of queries
        If $H$ query, $A$ chooses $x_i$ and is given $H(x_i)$
        If $F$ query, $A$ chooses $(s_i, c_i) \neq (s_0, c_0)$
            and is given $y_i = F_{p_0}(s_i, c_i)$
    until $A$ reaches the maximum number of queries
    $A$ outputs either 0 or 1
```

The success probability of A is defined as

$$Adv_A(t, m) = Pr_{E_1}[A = 1] - Pr_{E_0}[A = 1],$$

[2] If W is a set, than $w \overset{r}{\leftarrow} W$ denotes selecting w uniformly at random from W.

where t and m denote the maximum number of H and F queries, respectively. The maximum success probability achievable by any adversary A in is denoted by $Adv(t, m)$.

3 Password-Based KDFs in Practice

Password-based key derivation functions are commonly used in practice. They are also specified in industry standards such as PKCS#5, PKCS#12, IETF, and openPGP. Here we describe the KDFs in PKCS#5, which is considered as the de facto standard for password-based cryptography. KDFs in other standards mostly follow similar designs.

Two password-based KDFs are specified in PKCS#5 v2.0 [13]: PBKDF1 and PBKDF2. Some recommendations are given regarding the use of salt and iteration count. For example, S should be at least 64 bits, and c should be at least 1000.

The underlying function in PBKDF1 is a hash function $H()$ such as MD2, MD5, or SHA1. The derived key is defined as $y = H^{(c)}(p\|s)$. Most other standards or implementations use this construction.

PBKDF2 was intended to provide more security. The underlying function in PBKDF2 is a keyed hash function $H_k()$, such as HMAC [1]. The password p is used as the key k in each invocation of $H_k()$. The derived key is defined as $y = U_1 \oplus U_2 \oplus ... \oplus U_c$, where $U_i = H_p^{(i)}(s)$ for $i = 1, .., c$. The exclusive-ors adds an extra layer of protection, but at the core of the construction is still the iterative application of H_p.

4 Effects of Iteration Count

In this section, we focus our analysis on iteration count and quantify its effect on the security of KDF. A KDF function with an iterative structure is of the form

$$y = H^{(c)}(p, s) = H^{(c)}(p\|s).$$

We will show that this construction is secure as long as the adversary only has access to H and its computational resource is significantly less than $c|PW|$, which is formally stated in the following theorem.

Theorem 1. *In the weakly secure model for KDF, if the adversary makes at most t H queries, then the maximum success probability $Adv(t)$ satisfies*

$$\frac{\lfloor t/c_0 \rfloor}{|PW|} < Adv(t) < \frac{\lfloor t/c_0 \rfloor}{|PW|} + \frac{t^2}{2^n}.$$

Before going into the proof details, we first try to understand the result by considering a practical scenario. Let

$$|PW| = 2^{40}, n = 128, c = 2^{16}, t = 2^{44}.$$

Setting $c = 2^{16}$ adds little overhead at the user end for deriving a single key[3], but the workload of a straightforward dictionary attack increases to $c|PW| = 2^{56}$ from 2^{40} (when $c = 1$). With $t = 2^{44}$ queries to H, the attacker can certainly correctly compute a fraction of $\frac{t/c}{|PW|} = 2^{-12}$ of the derived keys. Our result shows that this is indeed the *best* the attacker can do, since the probability for correctly computing more than 2^{-12} of the derived keys is at most $\frac{t^2}{2^n} = 2^{-40}$. Effectively, the iteration count stretches a 40-bit password into a $40 + \log_2 c = 56$-bit key.

4.1 Graph Representation of H

For the purpose of the proof, we set up a graph to represent a random function H and the adversary's query process for H. This graph-based approach allows us to visualize the adversary's knowledge gained in the query process, and makes the proof more intuitive.

Let G_H be a directed graph on the vertex set $\{0,1\}^n$; a directed edge (x, y) exists in G_H if and only if $H(x) = y$. Hence every vertex has out-degree 1 and G_H contains 2^n edges. The adversary, by probing a sequence of t edge "$H(x) =?$", discovers a subgraph Q_H of G_H which is referred to as the *query graph*. Since the same query graph Q_H can arise from different functions H, it is sometimes convenient to write Q without referring to a specific H.

4.2 Analysis of Probabilities

We start by defining two games R (for "random") and K (for KDF) which correspond to the two experiments E_1 and E_0, respectively. For each game, we specify how to simulate the oracle H upon adversary's queries. In the game specification, there are some extra computing steps – they are hidden to A and hence do not affect the behavior of A, but they will help our analysis.

We note that the two games are very similar, and the only difference is in Step 4 which is shown by the underline. Two flags bad_1 and bad_2 are set when certain "bad" event occurs. The set Y contains all distinct values of $H(x)$ for which x has been queried[4].

In Game R, the answers seen by the adversary A are exactly the same as in E_1. The difference is that the game contains two extra steps – Step 2 for detecting collisions and Step 3 for detecting whether $H^{(c_0)}(p_0\|s_0)$ has been computed. So the success probabilities of A in game R and experiment E_1 are the same, which is stated in the following lemma.

Lemma 1. $Pr_R[A = 1] = Pr_{E_1}[A = 1]$.

In Game K, the answers seen by the adversary A are almost the same as in experiment E_0 with possible exception on queries $H(u_i)$. We will show in the next lemma that this apparent difference will not affect A's success probability.

[3] On a Pentium 4 running at 2.1GHz, 2^{16} SHA-1 operations take less than 0.02 second according to the benchmarks for Wei Dai's CRYPTO++ Library.

[4] We also include $u_0 = p_0\|s_0$ in Y is for detecting the event that u_0 is not the *first* vertex of a path. It is not necessary to do so, but makes later analysis easier.

```
Initially, $H(.)$ is undefined. Choose $p_0 \overset{r}{\leftarrow} PW$ and $y_0 \overset{r}{\leftarrow} \{0,1\}^n$.
Set $i \leftarrow 0, u_0 \leftarrow p_0 \| s_0, Y \leftarrow \{u_0, y_0\}$.

On oracle query $H(x)$:

  1. Choose $y \overset{r}{\leftarrow} \{0,1\}^n$.
  2. If $y \notin Y$, set $Y \leftarrow Y \cup \{y\}$.
     Else if $y \in Y$, set $bad_1$.
  3. If $x = u_i$ and $i < c_0$, set $i \leftarrow i+1$ and $u_i \leftarrow y$.
     Else if $x = u_i$ and $i = c_0$, set $bad_2$.
  4. Define $H(x) = y$ and return $y$.
```

Fig. 1. Game R.

```
Initially, $H(.)$ is undefined. Choose $p_0 \overset{r}{\leftarrow} PW$ and $y_0 \overset{r}{\leftarrow} \{0,1\}^n$.
Set $i \leftarrow 0, u_0 \leftarrow p_0 \| s_0, Y \leftarrow \{u_0, y_0\}$.

On oracle query $H(x)$:

  1. Choose $y \overset{r}{\leftarrow} \{0,1\}^n$.
  2. If $y \notin Y$, set $Y \leftarrow Y \cup \{y\}$.
     Else if $y \in Y$, set $bad_1$.
  3. If $x = u_i$ and $i < c_0$, set $i = i+1$ and $u_i \leftarrow y$.
     Else if $x = u_i$ and $i = c_0$, set $\underline{y \leftarrow y_0}$. Set $bad_2$.
  4. Define $H(x) = y$ and return $y$.
```

Fig. 2. Game K.

Lemma 2. $Pr_K[A = 1] = Pr_{E_0}[A = 1]$.

Proof: In experiment E_0, H is chosen randomly at the beginning and y_0 is then set to be $y_0 = H^{(c_0)}(u_0)$. Therefore, for $i = 0, 1, ...c_0 - 1$, each value $H(u_i)$ is chosen at random *before* the experiment starts. In Game K, for $i = 0, 1, ...c_0 - 2$, each value $H(u_i)$ is chosen at random as the game *proceeds*. Only the last value $H(u_{c_0 - 1})$ is chosen at random (to be y_0) *before* the game starts.

Since all these values are chosen at random and they are all independent of each other, there is no difference from the adversary's point of view. Hence the success probability of A is the same. **QED**

Using the above lemmas, we have $Adv_A(t) = Pr_R[A = 1] - Pr_K[A = 1]$. So we now consider the relation between Game R and Game K. Let BAD_1 be the event that flag bad_1 gets set, and similarly for BAD2. Let $\mathsf{BAD} = \mathsf{BAD}_1 \cup \mathsf{BAD}_2$. It is easy to see that the answers seen by A are exactly the same if neither bad event occurs. Furthermore, each bad event occurs with the same probability in the two games.

Lemma 3. *(1)* $Pr_R[A = 1|\overline{\mathsf{BAD}}] = Pr_K[A = 1|\overline{\mathsf{BAD}}]$.
(2) $Pr_R[\mathsf{BAD}] = Pr_K[\mathsf{BAD}]$.

Following a standard probability argument (such as that in [11]), we have $Adv_A(t) < Pr_R[\mathsf{BAD}]$. So we only need to derive an upper bound on $Pr_R[\mathsf{BAD}]$ for proving the theorem.

Proof of Theorem 1. For simplicity, we omit the R in the subscript.

$$\begin{aligned} Pr[\mathsf{BAD}] &= Pr[\mathsf{BAD}_1 \cup \mathsf{BAD}_2] \\ &= Pr[\mathsf{BAD}_1] + Pr[\mathsf{BAD}_2|\overline{\mathsf{BAD}_1}] \cdot Pr[\overline{\mathsf{BAD}_1}] \\ &\leq Pr[\mathsf{BAD}_1] + Pr[\mathsf{BAD}_2|\overline{\mathsf{BAD}_1}]. \end{aligned}$$

It is easy to see that $Pr[\mathsf{BAD}_1] < (t^2/2 + 2t)/2^n$, since the probability of a collision within t queries is at most $(t^2/2)/2^n$, and the probability that any of the t values of H collide with u_0 or y_0 is at most $2t/2^n$. Assuming $t \geq 4$, we have $Pr[\mathsf{BAD}_1] < t^2/2^n$.

We next bound the second term $Pr[\mathsf{BAD}_2|\overline{\mathsf{BAD}_1}]$. If the event BAD_1 doesn't occur, then the query graph consists of a set of disjoint paths on which u_0 can only appear as the *first* vertex. Note that BAD_2 is the event that there is a path of length at least c_0 starting from vertex u_0. With t edges, there are at most $\lfloor t/c_0 \rfloor$ such paths. For each path, with probability at most $1/|PW|$, the first vertex is $u_0 = p_0\|s_0$. Hence $Pr[\mathsf{BAD}_2|\overline{\mathsf{BAD}_1}] < \frac{\lfloor t/c_0 \rfloor}{|PW|}$. Combining the two upper bounds, we obtain that $Adv(t) < Pr[\mathsf{BAD}] \leq \frac{2t^2}{2^n} + \frac{\lfloor t/c_0 \rfloor}{|PW|}$.

The lower bound can be achieved easily by computing full paths of length c_0 for $\lfloor t/c_0 \rfloor$ passwords in PW. **QED**

4.3 Discussions on *c*-th Iterate of a Random Function

It may be helpful to review some mathematical background on the c-th iterate of a random function. Although random functions have been studied extensively in the literature, the c-th iterate function $H^{(c)}$ has received relatively little attention. For example, it is well known that the image of a random function has size $(1 - e^{-1})2^n$. What about the image of $H^{(c)}$? At what rate does the image size decrease with c? In a 1990 paper by Flajolet and Odlyzko [7] they provided answers to these questions by deriving the following recurrence.

Image Size of $H^{(c)}$. *The image size of $H^{(c)}$ is $(1 - \tau_c)2^n$, where τ_c satisfies the recurrence $\tau_0 = 0, \tau_{c+1} = e^{-1+\tau_c}$. Furthermore, asymptotically $1 - \tau_c = 2/c$.*

In other words, the image size of $H^{(c)}$ decreases arithmetically with c (not geometrically as one might guess at first). This illustrates that $H^{(c)}$ is significantly different from a random function as c gets larger, and its analysis is a nontrivial matter. It is also interesting to note that, although the image size of $H^{(c)}$ goes down by a factor of $2/c$ compared with that of H, yet Theorem 1 shows that the attacker's workload must increase by a factor of c.

Theorem 1 proves tight bounds for the password space. When $t < c$, the lower bound on $Adv(t)$ in the theorem becomes zero. Below, we derive a non-trivial lower bound by describing a strategy of the attacker. Let $d(x)$ denote the number of divisors of x, and define $d[x_1, x_2] = \max d(x), x_1 \leq x \leq x_2$.

Lemma 4. *With $t < c$ queries, the attacker can achieve $Adv_A(t) > \frac{t^2}{2^{n+1}} + (1 - \frac{t^2}{2^{n+1}})\frac{d[c,c-t]}{2^n}$.*

Proof. The attacker simply computes $u, H(u), H^{(2)}(u), \ldots$ iteratively and hopes that the sequence becomes periodic, i.e., a repeated value occurs before $H^{(t)}(u)$ so that $H^{(c)}(u)$ is determined. The probability for this to happen is $t(t+1)/2^{n+1}$. In the case the chain $(u, H(u), \ldots, H^{(t)}(u))$ is not periodic, the adversary will select a vertex on the path $v = H^{(\delta)}(u)$ where δ is chosen to maximize the number of divisors $d(c - \delta)$ for $0 \leq \delta \leq t$. The probability of success in this case is at least $d[c, c-t]/2^n$ as stated. **QED**

We note that, somewhat surprisingly, the lower bound gets better with larger c, as the attacker may find a number $c - \delta$ in the range $[c - t,\ c]$ with a large number of divisors so that it is more likely to have $H^{(c)}(u) = H^{(\delta)}(u)$ through periodicity.

In our modelling of the password space, we assumed that all passwords are of the same length ℓ and are chosen by users with equal probability. It is not difficult to extend our analysis to obtain similar security bounds when these assumptions are removed. For example, let a set of passwords of arbitrary length be added to PW. After one iteration of H these points are distributed randomly in the domain $\{0,1\}^n$, and additional iterations (adjusting c to be $c-1$) would behave just as analyzed before. Similarly, even if the passwords have different probabilities originally, after one iteration this will have no effect on the collision probabilities which depend only on the fact $prob[H(x) = y] = \frac{1}{2^n}$ in the domain $\{0,1\}^n$.

5 Security Analysis of KDFs in PKCS#5

In the preceding section, we analyzed the basic iterative construction $H^{(c)}$. The analysis implies that the two KDFs in PKCS#5 are weakly secure as long as the adversary's computational resource is far less than $c|PW|$, even though it can be much larger than $|PW|$.

In what follows, we analyze the security of the two KDFs under the strongly secure model – that is, the adversary is allowed a few queries to F. We show that neither KDF is secure under this model and we also explore how such security weakness can be exploited to launch attacks in practical settings.

5.1 PBKDF1

The attack on PBKDF1 is based on an obvious relation between keys derived using the *same* salt. For any salt s and two iteration counts $c_0 < c_1$, let $y_i = F(p, s, c_i) = H^{(c_i)}(p\|s)$. Then, it is easy to see that $y_1 = H^{(c_1 - c_0)}(y_0)$. This relation allows an attacker to distinguish y_0 from a random function with one F query (s, c_1) and $(c_1 - c_0)$ H queries.

Note that if the key $y_0 = H^{(c_0)}(p\|s)$ were ever compromised for some reason, then any key derived using the same salt s and an iteration count *larger than* c_0 would all be compromised. This might happen in practice if the user (or the security administrator of the system) decides to increment the iteration count. Therefore, it is a good practice in general to use different salt values in deriving different keys.

5.2 PBKDF2

The derived keys in PBKDF2 also suffer from non-randomness, although the relations among keys are slightly more complicated. Let s be any salt value and let c_1, c_2, c_3 be three consecutive iteration counts. For $i = 1, 2, 3$, define $y_i = F(p, s, c_i) = U_1 \oplus ... \oplus U_{c_i}$. Then, we have $y_1 \oplus y_2 = U_{c_2}$ and $y_2 \oplus y_3 = U_{c_3}$. This yields the following relation among the three keys: $(y_2 \oplus y_3) = H_p(y_1 \oplus y_2)$, where $H_p()$ is the underlying function HMAC.

This relationship among keys opens the door to dictionary attacks. The attacker simply computes the HMAC function $H_p(U_{c_2})$ for all possible passwords p, and the password that gives $H_p(U_{c_2}) = U_{c_3}$ is very likely to be the correct password used in the scheme. Once p is known, it is easy to distinguish the derived key from a random string. We remark that the workload of the above attack is $|PW|$, no matter what c is. This implies the iteration count does not add much (or any) protection in PBKDF2 against dictionary attacks.

6 Effects of Salt

A salt serves the purpose of creating a large set of possible long keys corresponding to a password p. If the salt is s bits long, then the number of possible long keys can be as large as 2^s. Each time the KDF is executed with a salt, either selected by the user or generated at random, one of the 2^s long keys is selected.

One natural question is the following: Suppose that an adversary has computed the long keys correspond to all the passwords $p \in PW$ for a salt s_1. That is, the adversary has a table of size $|PW|$ in which each entry contains the value $(p, H^{(c)}(p\|s_1))$ for some $p \in PW$. Does this table provide the adversary some shortcuts to derive long keys using a *different* salt $s_2 \neq s_1$?

The answer is certainly "No", which is well-known in practice. Using the graph-based approach, we can show that the set of paths corresponding to s_1 and the set of paths corresponding to s_2 are all disjoint with high probability, and hence the table for s_1 provides essentially no information for derived keys using s_2. The detailed analysis is similar to that for Theorem 1 and thus omitted here.

7 New Proposal for Strongly Secure KDF

In this section, we present a new proposal for strongly secure KDF based on our study on the effects of iteration counts and salt, as well as the analysis on existing KDFs.

We first note that the graph-based analysis provides insights on the exact way that each parameter contributes to the overall security: The computation process for deriving a key corresponds to a path in the query graph. So choosing a larger iteration count forces the attacker to traverse a *longer* path for deriving each key, while choosing a different salt value forces the attacker to traverse a *different* path in the computation. It is also easy to see why the KDFs in PKCS#5 are not strongly secure using the query graph. For example, in the case on PBKDF1, the F queries (s, c_0) and (s, c_1) correspond to two paths that overlap. This extra information allows the attacker to distinguish the derived key from a random string.

Based on the above discussion, we can see that a strongly secure KDF should be constructed in a way that the values of $y = F(p, s, c)$, for different p, s and c, are nearly independent of each other. Certainly, there are various ways of achieving this goal. Here we propose a simple construction that maintains the same efficiency as the KDFs in PKCS#5. The idea is to include iteration count explicitly as an input to the hash function H. More specifically, the new KDF is

$$y = F^*(p, s, c) = H^{(c)}(p\|s\|c).$$

In what follows, we prove that the above KDF is strongly secure – secure even when the adversary can choose (s, c) and make F queries. We assume that there are lower and upper limits to the c_i acceptable in queries to F, that is, $c_* < c_i < c^*$. Indeed, without a lower limit, the adversary can always set $c = 1$ in the F query and then perform an offline key search attack with complexity $O(|PW|)$.

Theorem 2. *In the strongly secure model for KDF, if the adversary makes at most t queries to H and at most m queries to F, then the maximum success probability $Adv(t, m)$ satisfies*

$$\frac{\max(\lfloor (t - c^*)/c_* \rfloor, m)}{|PW|} < Adv(t, m) \frac{\lfloor t/c_* \rfloor + 2m}{|PW|} + \frac{(t + m)^2}{2^n}.$$

Proof. We provide a sketch of the proof here, and the details are given in the appendix. The upper bound proof uses the same type of arguments as that of Theorem 1. More specifically, we define two games R' and K' which the adversary might play. Since there are now two types of queries to deal with, the simulator needs to maintain some extra information during the course of the game to make sure that its answers to oracle queries F^* and H are consistent. Then following a similar analysis, we only need to bound the probability of some bad events to obtain the upper bound in the theorem.

For the lower bound, we describe two strategies. In strategy A, the adversary computes separate paths of length c_* using queries to H and makes only one F query. In strategy B, the adversary constructs a single path of length t and picks m appropriate vertices to make m queries to F. The success probability is the maximum of the two as stated in the theorem. **QED**

8 Conclusions

Password-based key derivation functions are necessary in many security applications. Despite their importance and wide-spread usage, rigorous analysis of such functions seems to have received relatively little attention in the literature compared with many other cryptographic schemes.

In this paper, we define a general security framework for password-based key derivation functions where salt and iteration count are included as parameters. Under this framework, we focus on the most commonly used construction $H^{(c)}(p\|s)$ and prove that the iteration count c, when fixed, does have an effect of stretching the password by $\log_2 c$ bits. Our analysis is done using a random functional graph representing H, conditioned upon a query graph representing information revealed to the attacker. It provides insights on the exact way that each parameter contributes to the overall security.

We then analyze two widely deployed KDFs defined in PKCS#5. We show that both are secure the adversary cannot influence the parameters, but are subject to attacks otherwise. We also consider how such security weaknesses can be exploited in practice.

Finally, based on the insight gained from our earlier analysis, we propose a new password-based key derivation that is provably secure even when the attacker has full control of the salt and iteration count. The new proposal achieves stronger security while preserving the same efficiency as existing KDFs. We expect that the new proposal will find its application in practical implementations.

Acknowledgements

We would like to thank the anonymous referees for many helpful comments.

References

1. M. Bellare, R. Canetti and H. Krawczyk. Keyed Hash Functions for Message Authentication. In *Advances in Cryptology – Crypto '96*, Springer-Verlag, 1996. Crypto'96.
2. M. Bellare and P. Rogaway. Random Oracles are practical: A Paradigm For Designing Efficient Protocols. In *First ACM Conference on Computer and Communications Security*, 1993.
3. S. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 1992.
4. T. Dierks and C. Allen. The TLS Protocol Version 1.0. IETF RFC 2246, *Internet Request for Comments*, January 1999.
5. A. Hevia, A. Desai, and Y. L. Yin. A Practical-Oriented Treatment of Pseudorandom Number Generators. In *Advances in Cryptology – Eurocrypt '02*, Springer-Verlag, 2002.
6. FIPS PUB 186-2. Digital Signature Standard. National Institute of Standards and Technologies, 1994.

7. P. Flajolet and A. M. Odlyzko. Random mapping statistics. In *Advances in Cryptology - EUROCRYPT '89*, Springer-Verlag, 1990.
8. IEEE Std 1363-2000: Standard Specifications for Public-Key Cryptography. *IEEE Computer Society*, 2000.
9. IEEE P1363.2: Standard Specifications for Password-Based Public-Key Cryptographic Techniques. Draft D15. May 2004. http://grouper.ieee.org/groups/1363/passwdPK/draft.html.
10. J. Kelsey, B. Schneier, C. Hall, and D. Wagner. Secure Applications of Low-Entropy Keys. In *Proceedings of the First International Workshop ISW '97*, Springer-Verlag, 1998.
11. J. Killian and P. Rogaway. How To Protect DES Against Exhaustive Key Search Attacks. In *Advances in Cryptology - CRYPTO '96*, Springer-Verlag, 1996.
12. A. M. Odlyzko, private communication. 2003.
13. RSA Laboratories' PKCS#5 v2.0: Password-Based Cryptography Standard. 1999.
14. D. Wagner and I. Goldberg. Proofs of Security For The UNIX Password Hashing Algorithm. In *Advances in Cryptology - Asiacrypt '00*, Springer-Verlag, 2000.

Proof of Theorem 2

Upper Bound

The upper bound proof uses the same type of arguments as that of Theorem 1. We start by specifying two games R' and K' (see Figures 3 and 4). Since there are now both H and F^* queries to deal with, the simulator needs to maintain some necessary information during the course of the game to make sure that its answers to both types of oracle queries are consistent.

Before diving into the detailed descriptions of the two games, it is instructional to compare at a high level how oracle query H is handled in Game K' and Game K. The main difference is the additional Step 4 (marked as *new*) in Game K', which is for updating the necessary information maintained by the simulator.

In both games, the simulator keeps track of all the F^* queries as well as the H queries starting at $p_0\|s\|c$. More precisely, it maintains a set

$$L = \{(s_k, c_k, y_k, u_k, i_k)\}$$

where each item in L is a 5-tuple such that either the query (s_k, c_k) has been made to F^* or the query $x = p_0\|s_k\|c_k$ has been made to H. The other three entries are defined as follows:

– If the query to F^* has been made, then $y_k = H^{(c_k)}(x)$ Otherwise, $y_k = *$ meaning it is still undefined.
– If the query to H has been made, then i_k is the number of consecutive queries to H made thus far starting at x, and u_k is the last answer.

It also maintains the set of all "starting points" in L, that is, a set $X = \{x_k\}$ where $x_k = p_0\|s_k\|c_k$.

Following similar analysis as that of Theorem 1, we have that $Adv_A(t, m) < Pr[\mathsf{BAD}] \le Pr[\mathsf{BAD}_1] + Pr[\mathsf{BAD}_2|\overline{\mathsf{BAD}_1}]$. So we only need to derive an upper

```
Initially, $H(.)$ and $F_{p_0}(.,.)$ are both undefined.
Choose $p_0 \xleftarrow{r} PW$ and $y_0 \xleftarrow{r} \{0,1\}^n$.
Set $i_0 \leftarrow 0, x_0 \leftarrow p_0\|s_0\|c_0, Y \leftarrow \{y_0\}$.
Set $X \leftarrow \{x_0\}, L \leftarrow \{(s_0, c_0, y_0, u_0, i_0)\}$.
Set $j \leftarrow 0$.

On oracle query $H(x)$:

1. Choose $y \xleftarrow{r} \{0,1\}^n$.
2. If $y \notin Y$, set $Y \leftarrow Y \cup \{y\}$.
   Else if $y \in Y$, set $bad_1$.
3. If $x = x_k \in X$ and $i_k < c_k$, set $i_k \leftarrow i_k + 1$ and $u_k \leftarrow y$.
   Else if $x = x_k \in X$ and $i_k = c_k$, set $\underline{y \leftarrow y_k}$. Set $bad_2$.
4. (new step compared with Game K)
   If $x \notin X$ and $x = p_0\|s\|c$, then $X \leftarrow X \cup x$ and add a new item in $L$:
   $j \leftarrow j+1, s_j \leftarrow s, c_j \leftarrow c, y_j \leftarrow *, u_j \leftarrow x, i_j \leftarrow 1$
5. Define $H(x) = y$ and return $y$.

On oracle query $F^*_{p_0}(s, c)$:

1. Choose $y \xleftarrow{r} \{0,1\}^n$.
2. If $y \notin Y$, set $Y \leftarrow Y \cup \{y\}$.
   Else if $y \in Y$, set $bad_1$.
3. Let $x = p_0\|s\|c$.
   If $x = x_k \in X$ and $i_k < c_k$, set $y_k \leftarrow y$.
   Else if $x = x_k \in X$ and $i_k = c_k$, set $\underline{y \leftarrow y_k}$. Set $bad_2$.
4. If $x \notin X$, then $X = X \cup x$ and add a new item in $L$:
   $j \leftarrow j+1, s_j \leftarrow s, c_j \leftarrow c, y_j \leftarrow y, u_j \leftarrow x, i_j \leftarrow 0$
5. Define $F^*_{p_0}(s, c) = y$ and return $y$.
```

Fig. 3. Game K'.

Game R' is the same as Game K', except that the execution of the underlined step ($y \leftarrow y_k$) is removed.

Fig. 4. Game R'.

bound on the probability of each bad event. Analyzing the first term is straightforward. Since there are $t + m$ queries in total, $Pr[\mathsf{BAD}_1] < (t + m)^2/2^n$.

Analyzing the second term $Pr[\mathsf{BAD}_2|\overline{\mathsf{BAD}_1}]$ is somewhat more complex. First, let Q_1 be the query graph corresponding to the t H queries. If the attacker uses only Q_1, its success probability is bounded by $\frac{\lfloor t/c_* \rfloor}{|PW|}$ as shown in Theorem 1, except that c_0 is replaced with its lower limit c_*. Next, we consider the effect of F^* queries. We observe that (unlike in the proof of Theorem 1) $x_j = p_0\|s_j\|c_j$ doesn't have to be the *first* vertex of a path, since the adversary is allowed to choose (s_k, c_k) and make a query to F^*, and this provides the adversary more chances of success. To quantify this advantage, we consider the number of vertices of the form $\{p\|s_i\|c_i, 1 \leq i \leq m\}$, denoted by m'. The success probability using F^* queries is bounded by $m'/|PW|$. Note that $m' = m + q$ where q is the expected number of collisions in $s\|c$ among all the vertices $p\|s\|c$ in Q_1. Since the expected value of q is $t|PW|/2^n << 1$, it can be shown that the probability that $m' = m + q \geq m + m = 2m$ is negligible.

Combining all the probabilities, we prove that $Adv(t, m)$ is bounded by $\frac{\lfloor t/c_* \rfloor + 2m}{|PW|} + \frac{(t+m)^2)}{2^n})$ as stated.

Lower Bound

For the lower bound, we describe two strategies from which the adversary can pick the one yielding better success probability depending on the parameters. In strategy A, the adversary computes separate paths of length c_* for $\lfloor (t - c^*)/c_* \rfloor$ passwords $p_i\|s\|c_*$ using $t - c^*$ queries to H. He then makes a F query asking for $y = F^*_p(s, c_*)$. With probability $\lfloor (t - c^*)/c_* \rfloor / |PW|$, vertex y coincides with the endpoint of one of the paths, thus revealing the password p_0. In such an event the adversary then makes c_0 more H queries to compute $y'_0 = H^{(c_0)}(p_0\|s_0\|c_0)$ and answers 1 if $y'_0 = y_0$. All together the adversary used at most t queries to H and one F queries to achieve success probability of $\lfloor (t - c^*)/c_* \rfloor / |PW|$.

In strategy B, the adversary constructs Q_1 to be a single path of length t starting from an arbitrary $p\|s\|c$. With probability $1 - O(t^2/2^n)$, the path will be cycle-free. Its first $t - c^*$ vertices $p_i\|s_i\|c_i$ have their full paths $T_{p_i\|s_i\|c_i}$ completely contained in Q_1. Assuming m to be much smaller than $t - c^*$, the adversary can pick m vertices $p_i\|s_i\|c_i$ along the path with distinct p_i and make at most m queries to F with the corresponding (s_i, c_i)'s. With probability $m/|PW|$, it can identify the password. This completes the proof of Theorem 2. **QED**