The M3lcrypt Password Based Key Derivation Function

Isaiah Makwakwa
imakwakwa@gmail.com

Abstract

M3lcrypt (canonical M3lcrypt$_H$) is a password based key derivation function built around the Merkle-Damgard hash function H. It uses large [pseudo]random salt values ($\geq$ 128-bit) and supports password lengths asymptotically close to the maximum allowable input size for H.

1 Introduction

The user induced probability distribution, χ_u, on the password space, PW (e.g a subset of the 95 printable 7-bit ASCII characters) has inherent low entropy [7, 11, 10]. Therefore, PW as a source of cryptographic key material is vulnerable to both *brute force* and *dictionary* attacks [6, 13, 5, 8].

Definition 1.1. *A password-based key derivation function is a pseudorandom function family, $\{PBKDF_s\}_{s\in S}$, such that for any salt value $s \in S$ there exists a pseudorandom function,*

$$PBKDF_s : PW \rightarrow \{0,1\}^L,$$

that maps from χ_u into a distribution suitable for direct cryptographic use.

A *pseudorandom function family* is necessary, for since χ_u has low entropy, a deterministic function as opposed to a pseudorandom function family, allows precomputation of tables of cryptovariables for all or part of the password space. Thus, it facilitates on the fly computation of pre-images.

However, high functional dependency on auxiliary randomness (often non-secret), $s \in S$, from a large space increases the uncertainty associated with the PBKDF. Indeed, with a large salt space, the space-time complexity for complete or partial precomputation might be out of reach of even the most well resourced of adversaries [5, 6, 8].

For *a distribution suitable for direct cryptographic use*, we assume computational (pseudorandom) rather than statistical closeness to uniform. It suffices, therefore, to show that within feasible computational effort, the adversary's distinguishing probability will remain below a certain [small] threshold [13, 3]. In particular, we require this to hold even with adversarial control or knowledge of part of the cryptovariable [4].

In turn, this may require certain assumptions on the underlying cryptographic primitives such as the idealised assumptions of the random oracle model, the combinatorial arguments of the universality of hash functions and equivalent assumptions for symmetric key ciphers [4].

The expectation being that, apart from iterative calls to the PBKDF on password dictionaries, taking pre-images and/or partial information on the password space should be computationally infeasible [2].

More subtly, however, entropy and complexity requirements on key material for practical cryptosystems make certain elements of *key stretching* such as the iterative application of some cryptographic primitives necessary. For example, under the idealised oracle model, the above conditions would imply that a hash of the password and some function of the salt value can be considered within certain computational parameters to be a secure PBKDF. Indeed many password hashing implementations follow this approach.

However, in general, this lowers the complexity of key search attacks below that designed for practical cryptosystems. On the other hand, it is known that key stretching techniques, in the absence of design flaws such as narrow pipes and reusable internal values, allow for a quantifiable increase in the complexity of dictionary and brute force attacks [6, 13].

The above not withstanding, a moderately resourced adversary may build special purpose key search machines (e.g. the Electronic Frontier Foundation's DES cracker [8], M. Wiener's design for a DES cracker [10]) that dramatically reduce the [area-time] cost of brute force attacks.

Indeed, by Moore's law [10], [cryptographic] circuits not only become faster but cheaper and smaller allowing for greater parallelism and dramatic growth in the economies of scale available to the attacker. Therefore, hardware-frustrating techniques such as memory and/or expensive operations may be necessary for imposing cost constraints on custom circuits while ensuring efficiency of computation on general purpose processors [6, 2].

In this paper a new password based key derivation function, M3lcrypt, with the canonical name M3lcrypt$_H$ is proposed. M3lcrypt$_H$ is based on the iterative application of a PRF construct based on the Merkle-Damgard function H. The paper is organised as follows. Section 2 provides a detailed specification of the scheme, security goals and usage scenarios are discussed in Section 3, design choices are provided in Section 4, Section 5 provides an initial security analysis, Section 6 motivates some efficiency considerations and a conclusion is given in Section 7.

2 The M3lcrypt$_H$ PBKDF

Let $h : \{0,1\}^m \times \{IV\} \rightarrow \{0,1\}^n$ be the compression function for the Merkle-Damgard hash function H with initialisation vector IV, block length m and digest size n.

Definition 2.1. *A compression function family, $\{h_k\}_{k\in\{0,1\}^n}$, is a function family, such that for every $k \in \{0,1\}^n$, the compression function*

$$h_k : \{0,1\}^m \times \{k\} \to \{0,1\}^n$$

is derived from h by setting $IV = k$.

We denote by H_k the Merkle-Damgard hash function based on h_k [4].

Let $M, N, c, L > 0 \in \mathbb{Z}$, $p \in PW$, $s \in S = \{0,1\}^w$, $V, X \subset \{0,1\}^n$ and PRF_k be a pseudorandom function family based on $\{h_k\}_k$. The M3lcrypt$_H$ PBKDF, $F_{N,M}$, is the algorithm:

Algorithm: $\mathbf{F}_{N,M}$(p,s,c,L)

require: $w \geq 128$, $N \geq 2^{15}$, $M \geq 2^4$ and $c \geq 2^{13}$.

$l \leftarrow \lceil \frac{L}{n} \rceil$
$r \leftarrow l * n - L$
$digest \leftarrow$ s
for $i = 0$ **to** $N - 1$ **do**
    digest $\leftarrow PRF_p(digest||c||i)$
    X[i] $\leftarrow$ digest
end for
$shash \leftarrow PRF_s(p||digest)$
for $i = 0$ **to** $c - 1$ **do**
    k $\leftarrow$ digest % N
    digest $\leftarrow PRF_{shash}(X[k]||digest||i)$
    V[i % M] $\leftarrow$ digest
end for
$shash \leftarrow PRF_{shash}(p||s||digest||c)$
$thash \leftarrow V[0]||V[1]||\cdots||V[M-1]$
$y \leftarrow$ *empty string*
for $i = 0$ **to** $l - 1$ **do**
    digest $\leftarrow PRF_{shash}(p||digest||thash)$
    if $i = l - 1$ **and** $r! = 0$ **then**
        y $\leftarrow y||digest_{|r}$
    else
        y $\leftarrow y||digest$
    end if
end for
return y

where $x_{|r}$ is the first r bits of x and PRF is instantiated with HMAC. Note that the above *requirements* have been designed and tested for SHA2 only.

3 Design Goals

The following is a brief summary of the design goals for M3lcrypt$_H$.

Pseudorandomness: an output distribution indistinguishable from random by adversaries with oracle access to PRF even with partial control or knowledge of the cryptovariable.

Dual use: usable for both password hashing and cryptographic key derivation.

4 Design Rationale

We list and motivate some of the M3lcrypt$_H$ design choices as follows.

4.1 Use of HMAC as PRF

Pseudorandomness: preserve randomness, independence and unpredictability of output even with adversarial knowledge or control of part of the cryptovariable.

Pre-image resistance: ensure resistance to non-trivial pre-image attacks (conditional on instantiation with a secure hash function).

Random oracle modeling: admit random oracle modeling for PRF via random oracle domain extension on the compression function, h.

4.2 Configurable Size Array X

Imposes memory constraints on any $F_{N,M}$-computing circuits by the requirement of random access to X. Therefore, N determines the amount of random access memory required to compute the cryptovariable and thus, the cost of custom circuits.

4.3 Configurable Size Array V

Contains sufficient randomness per computation to effectively contribute to the entropy of the cryptovariable. In particular, it reduces the incidence of narrow pipes as well as short cut and *meet-in-the-middle* attacks. Specifically, given initial outputs, it reduces the possibility of *meet-in-the-middle* attacks in the computation of subsequent cryptovariables (i.e. using *lines 20-27* only).

4.4 Mixed PRF Modes (*Lines 8, 14 and 21*)

Counter mode: reduces the possibility of rapid iteration through reuse of internal state (such as collision induced cycles) by forcing different inputs into each successive call to PRF.

Feedback mode: provides non-adversarial controlled input randomisation which contributes to the randomisation properties. It also reduces the impact of parallelism by increasing dependency between successive hashing steps.

5 Security Analysis

5.1 Security Model

Our security model assumes the adversary is a polynomial-time algorithm A with knowledge of $N, M, c \in \mathbb{Z}$, $s \in S$ and which can make any number of [PRF] oracle queries in distinguishing the PBKDF output from a random bit string of the same length. The number of oracle queries the adversary can feasibly make constitutes its computational power [13, 4, 3].

Random oracle modeling for PRF is motivated by adversarial difference to structural over PRF attacks (which in the specific case of HMAC reduces to hash function attacks [1, 12]). In any case, many PRFs (including HMAC) support module drop in which reduces the impact of attacks on individual hash functions [1, 13].

For security, we demand that within feasible computational effort, the adversary's distinguishing probability will remain below a certain [small] threshold even with adversarial control or knowledge of part of the cryptovariable (Definition 1.1).

For brevity, we limit our analysis to the n-bit output, $y = F_{N,M}(p, s, c, n)$, since the inclusion of *shash* and V in the output transformation at *line 21* ensures equivalent analysis for further outputs.

We assume, therefore, that A has oracle access to PRF that on input (k, x) returns $\mathrm{PRF}_k(x)$ and that A does not repeat an oracle query. Effectively, A mirrors *Experiment* E_b in Figure 1 [9, 13].

Essentially, A guesses how y_0 was arrived at by making queries to PRF. It outputs a 1 to guess $F_{N,M}$ or 0 otherwise. A wins the game if it guesses correctly with probability greater than 0.5.

The adversary's *advantage*, denoted Adv_A, is its probability of guessing right, normalised to a $[-1, 1]$ scale where -1 indicates the strategy of always guessing wrong and 1 indicates the strategy of always guessing correct. Guessing at random or always guessing the same way will give an advantage of 0.

Therefore, A's success probability is defined by

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

where t denotes the maximum number of queries to PRF [3, 9]. The maximum success probability achievable by any adversary A is denoted $Adv(t)$.

On the other hand, the requirement of random oracle modeling for PRF imposes constraints on our choices for PRF. For example, constructs vulnerable to extension attacks (such a the key-prepend construct) may not be used for PRF. Fortunately, the HMAC construction admits random oracle modeling through random oracle domain extension for the compression function [4].

Therefore, the rest of the document assumes random oracle modeling for PRF and may invariably refer to PRF as the random oracle.

Figure 1: Experiment E_b

$b \xleftarrow{r} \{0,1\}$ // Bit b is selected at random // independent of A

$p_0 \xleftarrow{\chi_u} PW$ // Password is selected // according to distribution χ_u

$h \xleftarrow{r} \{f \mid f : \{0,1\}^m \rightarrow \{0,1\}^n\}$ // Compression function // is generated at random

$s_0 \leftarrow S$ // Known fixed value

$c_0 \leftarrow \mathbb{Z}, N \leftarrow \mathbb{Z}, M \leftarrow \mathbb{Z}$ // Known fixed integers

if $b = 0$ **then**

  $y_0 \leftarrow F_{N,M}(p_0, s_0, c_0, L)$

else

  $y_0 \xleftarrow{r} \{0,1\}^n$ // y_0 is selected at random

$i \leftarrow 0$

repeat

  $i \leftarrow i + 1$

  A chooses k and x_i. It then makes the oracle query (k, x_i), receiving the oracle response $PRF_k(x_i)$. This process repeats until the maximum no. of queries is reached.

A outputs either 0 or 1

Figure 1: Experiment E_b

5.2 Analysis of Probabilities

Consider Game R and Game K in Figure 2 and Figure 3 respectively.

Note that Game R and Game K are similar except for the underlined step in Game K. Further, note that flag bad_1 signals *all* instances of collisions in oracle responses and not just those within PRF_k for fixed k. Flag bad_2 is used to signal that $y_0 = F_{N,M}(p_0, s_0, c_0, n)$ has been computed.

The set Y is initialised with y_0 to ensure that collisions with this value are properly signaled.

Let BAD_1 be the event that the flag bad_1 and BAD_2 be the event that the flag bad_2 get set respectively. Define $BAD = BAD_1 \cup BAD_2$, the event that either flag is set.

Claim 5.1. $Pr_R[A = 1] = Pr_{E_1}[A = 1]$

Claim 5.2. $Pr_K[A = 1] = Pr_{E_0}[A = 1]$

Claim 5.3. $Pr_R[A = 1 \mid \overline{BAD}] = Pr_K[A = 1 \mid \overline{BAD}]$

Claim 5.4. $Pr_R[BAD] = Pr_K[BAD]$

Proof. Same as arguments of Lemma 1, 2 and 3 in [13] □

Therefore, $Adv_A(t) = Pr_K[A = 1] - Pr_R[A = 1]$ and thus, by Claim 3.5 in [9], $Adv_A(t) < Pr_R[BAD]$.

Lemma 5.1. *Let $\mathcal{H}(PW)$ be the entropy of PW with respect to χ_u, then*

$$\frac{\lfloor t/(c_0+N+3)\rfloor}{2^{\mathcal{H}(PW)}} \leq Adv_A(t) < \frac{t^2+2t}{2^n} + \frac{\lfloor t/(c_0+N+3)\rfloor}{2^{\mathcal{H}(PW)}}.$$

Proof. Without loss of generality, we may assume the brute force attacker treats PW as a uniformly distributed $\mathcal{H}(PW)$-bit entropy source [10, 6, 4].

$$\begin{aligned} Pr_R[BAD] &= Pr_R[BAD_1 \cup BAD_2] \\ &= Pr_R[BAD_1] + Pr_R[BAD_2 \mid \overline{BAD_1}] \\ &\quad \times Pr_R[\overline{BAD_1}] \\ &\leq Pr_R[BAD_1] + Pr_R[BAD_2 \mid \overline{BAD_1}] \end{aligned}$$

Clearly, since the probability of collisions among t queries is upper bounded by $\frac{t(t+1)}{2^{n+1}} < \frac{t^2+t}{2^n}$ and that of collisions with y_0 by $\frac{t}{2^n}$, $Pr_R[BAD_1] < \frac{t^2+2t}{2^n}$.

On the other hand, if event BAD_1 does not occur, oracle responses form $\lfloor \frac{t}{c_0+N+3} \rfloor$ disjoint sets of length $c_0 + N + 3$. The probability that any one of these sets coincides with the set generated in the computation of y_0 from p_0 is $\frac{\lfloor \frac{t}{c_0+N+3} \rfloor}{2^{\mathcal{H}(PW)}}$.

Therefore, $Pr_R[BAD_2 \mid \overline{BAD_1}] \leq \frac{\lfloor t/(c_0+N+3)\rfloor}{2^{\mathcal{H}(PW)}}$, proving

$$Adv_A(t) < \frac{t^2+2t}{2^n} + \frac{\lfloor t/(c_0+N+3)\rfloor}{2^{\mathcal{H}(PW)}}.$$

The lower bound is achieved by generating $\lfloor \frac{t}{c_0+N+3} \rfloor$ disjoint sets of length $c_0 + N + 3$ corresponding to $\lfloor \frac{t}{c_0+N+3} \rfloor$ distinct passwords. □

We make the following observation concerning the inclusion of digest length n in the expression for $Adv_A(t)$. If $\mathcal{H}(PW) > 2n$, which is not inconceivable in practice (e.g. long FIPS 181 compliant passwords [11]), the bound is not tight. However, substituting for hash functions with longer digest lengths may substantially reduce the impact of increasing $Pr_R[BAD_1]$ values.

Moreover, Section 5.3 shows that whenever $\mathcal{H}(PW) \geq 5n$ there exists a meet-in-the-middle attack with complexity less than exhaustive search.

The foregoing notwithstanding, and assuming security for n-bit hash functions, Lemma 5.1 shows that password security derives from the user induced distribution χ_u. In short, within the confines of our security model, $F_{N,M}$ is as "secure as the passwords users choose" [8].

Figure 2: Game R

$PRF_{(.)}(.)$ is undefined.
Choose $p_0 \xleftarrow{X_u} PW$ and $y_0 \xleftarrow{r} \{0,1\}^n$.
Set $V \leftarrow [\,]$; $X \leftarrow [\,]$; $digest \leftarrow s_0$; $i \leftarrow 0$; $j \leftarrow 0$; $sh \leftarrow$ *empty string*
$Y \leftarrow \{y_0\}$.

On oracle query $PRF_{(k)}(x)$

1. **Choose** $y \xleftarrow{r} \{0,1\}^n$
   if $y \in Y$ **then**
   set bad_1
   else
   $Y \leftarrow Y \cup \{y\}$
   end if
2. **if** $k = p_0$ **and** $i < N-1$ **and** $j = 0$ **then**
   if $x = (digest||c_0||i)$ **then**
   $digest \leftarrow y$; $X[i] \leftarrow y$; $i \leftarrow i+1$
   end if
3. **else if** $k = s_0$ **and** $i = N-1$ **and** $j = 0$ **then**
   if $x = (p_0||digest)$ **then**
   $sh \leftarrow y$; $j \leftarrow j+1$
   end if
4. **else if** $k = sh$ **and** $i = N-1$ **and** $j > 0$ **and** $j < c_0$ **then**
   if $x = (X[digest \,\%\, N]||digest||j)$ **then**
   $V[j \,\%\, M] \leftarrow y$; $digest \leftarrow y$; $j \leftarrow j+1$
   end if
5. **else if** $k = sh$ **and** $i = N-1$ **and** $j = c_0$ **then**
   if $x = (p_0||s_0||digest||c_0)$ **then**
   $sh \leftarrow y$; $j \leftarrow j+1$
   end if
6. **else if** $k = sh$ **and** $i = N-1$ **and** $j = c_0+1$ **then**
   $thash \leftarrow V[0]||V[1]||V[2]||\cdots||V[M-1]$
   if $x = (p_0||digest||thash)$ **then**
   set bad_2
   end if
   end if
7. **define** $PRF_{(k)}(x) = y$ **and return** y

Figure 2: Game R

Figure 3: Game K

Initial declarations just as in Game R.

On oracle query $\mathrm{PRF}_{(k)}(x)$

1. **Choose** $y \xleftarrow{r} \{0,1\}^n$
   if $y \in Y$ **then**
   set bad_1
   else
   $Y \leftarrow Y \cup \{y\}$
   end if
2. **if** $k = p_0$ **and** $i < N-1$ **and** $j = 0$ **then**
   if $x = (digest||c_0||i)$ **then**
   $digest \leftarrow y$; $X[i] \leftarrow y$; $i \leftarrow i+1$
   end if
3. **else if** $k = s_0$ **and** $i = N-1$ **and** $j = 0$ **then**
   if $x = (p_0||digest)$ **then**
   $sh \leftarrow y$; $j \leftarrow j+1$
   end if
4. **else if** $k = sh$ **and** $i = N-1$ **and** $j > 0$ **and** $j < c_0$ **then**
   if $x = (X[digest \,\%\, N]||digest||j)$ **then**
   $V[j \,\%\, M] \leftarrow y$; $digest \leftarrow y$; $j \leftarrow j+1$
   end if
5. **else if** $k = sh$ **and** $i = N-1$ **and** $j = c_0$ **then**
   if $x = (p_0||s_0||digest||c_0)$ **then**
   $sh \leftarrow y$; $j \leftarrow j+1$
   end if
6. **else if** $k = sh$ **and** $i = N-1$ **and** $j = c_0+1$ **then**
   $thash \leftarrow V[0]||V[1]||V[2]||\cdots||V[M-1]$
   if $x = (p_0||digest||thash)$ **then**
   set $\underline{y = y_0}$; set bad_2
   end if
   end if
7. **define** $\mathrm{PRF}_{(k)}(x) = y$ **and return** y

Figure 3: Game K

5.3 Meet-in-the-middle Attack

The following meet-in-the-middle attack based on [4] recovers the password with effort less than exhaustive search whenever $\mathcal{H}(PW) \geq 5n$.

We assume that PW consists of strings of length greater than the block length of the underlying hash function. For brevity, we also assume that length

appending in hash computations is omitted (the attack holds in either case).

Let PW be an m-entropy source ($m \geq 5n$) and assume adversarial knowledge of, $y = y_0 y_1 \cdots y_{m-1}$ ($|y_i| = n$, $\forall i$), the $F_{N,M}$ output for $p \in PW$.

1. Find $m_1, m_2 \in \mathbb{Z}$ such that PW can be expressed as the cartesian product of sources M_1 and M_2 of entropy m_1 and m_2 respectively.
2. For each $p_2 \in M_2$ and $K_1, K_2, K_3, K_4 \in \{0,1\}^n$
   (a.) Compute
   $$k \leftarrow H_{K_1}(p_2).$$
   (b.) For $0 \leq i < N$, compute
   $$\begin{aligned} digest &\leftarrow HMAC_k(digest||c||i), \\ X[i] &\leftarrow digest. \end{aligned}$$
   (c.) Compute
   $$shash_0 \leftarrow H_{IV}\left(s^+ \oplus opad||H_{K_2}\left(p_2||digest\right)\right),$$
   where x^+ is the string x zero padded to block length and *opad* is the HMAC binary constant.
   (d.) For $0 \leq i < c$, compute
   $$\begin{aligned} digest &\leftarrow HMAC_{shash_0}(X[digest \,\%\, N]||digest||i), \\ V[i \,\%\, M] &\leftarrow digest. \end{aligned}$$
   (e.) Compute
   $$shash_1 \leftarrow H_{IV}\left(shash_0^+ \oplus opad||H_{K_3}\left(p_2||s||digest||c\right)\right).$$
   (f.) Finally, compute
   $$y_0^{'} \leftarrow H_{IV}\left(shash_1^+ \oplus opad||H_{K_4}\left(p_2||digest||thash\right)\right),$$
   where $thash = V[0]||V[1]||\cdots||V[M-1]$.
   (g.) Create a table T of values
   $$(K_1, K_2, K_3, K_4, shash_0, shash_1, p_2)$$
   for which $y_0 = y_0^{'}$.
3. For each $p_1 \in M_1$
   (a.) Check if the tuple
   $$(H_{IV}(p_1), H_{IV}(s^+ \oplus ipad||p_1), K_3, K_4, shash_0, shash_1, p_2)$$
   for some $K_3, K_4, shash_0, shash_1, p_2$ are in T. If so,
   (b.) Check if

$$K_3 = H_{IV}\left(shash_0^+ \oplus ipad||p_1\right).$$

where *ipad* is the HMAC binary constant.

(c.) If (b.) holds, check if

$$K_4 = H_{IV}\left(shash_1^+ \oplus ipad||p_1\right).$$

(d.) If (c.) holds, check if $p^{'} = p_1||p_2$ [under $F_{N,M}$] produces the substring $y_1 y_2 \cdots y_{m-1}$ of the string y (corresponding to p). If so, output $p = p_1||p_2$.

For a random hash function the probability of producing the string $y = y_0 y_1 y_2 \cdots y_{m-1}$ is 2^{-m}. Therefore, with high probability a single password p will be returned in step 3(d.).

Assuming the hash function can be computed in unit time, the complexity of the attack is as follows. We require time $2^{4n+m_2+log_2 c+log_2 N+1}$ for step 2. Since $|T| = 2^{3n+m_2}$ (there are 2^{4n+m_2} possible tuples,

$$(K_1, K_2, K_3, K_4, shash_0, shash_1, p_2)$$

and y_0 can be produced with prob. 2^{-n} for random H), sorting T for step 3. requires negligible time relative to $2^{4n+m_2+log_2 c+log_2 N+1}$. Finally, each of the 2^{m_1} checks in step 3(a.) requires a table lookup with time complexity of order $log_2|T| = 3n + m_2$.

Therefore, all together the attack requires time $2^{4n+m_2+log_2 c+log_2 N+1}+2^{m_1}(3n+m_2)$. For example, for $m = 5n$, setting $m_1 = 4.5n$ and $m_2 = 0.5n$ we get complexity $\approx 2^{4.5n+log_2 c+log_2 N+2}$ less than the 2^{5n} required for exhaustive search.

However, the attack can be made prohibitive relatively easily. Consider the effect of substituting

$$digest \leftarrow PRF_{digest}(p||c||i).$$

for *line 8* of the algorithm. Clearly, the above attack now requires effort of the order $2^{4n+Nn+m_2+log_2 c+log_2 N+1}+2^{m_1}(3n+Nn+m_2)$ which is all but practically out of reach of even the most well resourced of adversaries even for relatively small values of N (assuming enough storage for intermediate values).

On the other hand, the significant computation time differences among passwords with varying lengths outweighs the perceived benefits (esp. when n-bit hashes are considered secure).

5.4 Effect of V

Suppose the output step in $F_{N,M}$ (*line 21*) was replaced by

$$digest \leftarrow PRF_{shash}(p||digest),$$

then the computation of *shash* at *line 17* represents a narrow pipe.

Therefore, an adversary with oracle access to an oracle that guesses the value of *shash* at *line 17* with significant probability can compute the final

cryptovariable with much less complexity. However, on average, by our idealised assumptions on PRF the adversary is liable to expending more effort ($\approx 2^{n+\mathcal{H}(PW)}$) than in a straight forward exhaustive search.

However, assuming that PW consists of strings of length greater than block length and $\mathcal{H}(PW) \geq 5n$, the adversary can do better. The following meet-in-the-middle attack based on [4] can recover the password in time less than exhaustive search.

In short, the adversary tries to compute successive outputs (e.g. y_1 the lvalue of 2(a.)) based on partially known values (e.g. y_0 in the rvalue of 2(a.)) using *line 21* only.

Let $\mathcal{H}(PW) = m \geq 5n$ and $y = y_0 y_1 y_2 \cdots y_m$ ($|y_i| = n$, $\forall i$) be the initial output of $F_{N,M}$ for $p \in PW$.

1. Let $m_1, m_2 \in \mathbb{Z}$ be such that PW can be expressed as the cartesian product of sources M_1 and M_2 of entropy m_1 and m_2 respectively.
2. For each $p_2 \in M_2$ and $K_1, K_2 \in \{0,1\}^n$
   (a.) Compute
   $$y_1^{'} = H_{IV}\left(K_1^+ \oplus opad || H_{K_2}\left(p_2 || y_0\right)\right).$$
   (b.) Create a table T of values
   $$(K_1, K_2, p_2)$$
   for which $y_1 = y_1^{'}$.
3. For each $p_1 \in M_1$ and $K_3 \in \{0,1\}^n$
   (a.) Check if the tuple
   $$\left(K_3, H_{IV}(K_3^+ \oplus ipad || p_1), p_2\right)$$
   for some p_2 is in T.
   (b.) If (a.) holds, use *line 21* of the algorithm (and values y_i) to check if $p^{'} = p_1 || p_2$ produces the substring $y_2 y_3 \cdots y_m$ of the string y (corresponding to p). If so, output $p = p_1 || p_2$.

For a random hash function the probability of producing the substring $y_1 y_2 \cdots y_m$ of y is 2^{-m}. Therefore, with high probability a single password p will be returned in step 3(b.).

The complexity of the attack is as follows. We require time 2^{2n+m_2} for step 2. Since $|T| = 2^{n+m_2}$ (there are 2^{2n+m_2} possible tuples, (K_1, K_2, p_2) and y_1 can be produced with prob. 2^{-n} for random H), sorting T for step 3. requires negligible time relative to 2^{2n+m_2}. Finally, each of the 2^{n+m_1} checks in step 3(a.) requires a table lookup with time complexity of order $log_2|T| = n + m_2$.

Therefore, all together the attack requires time $2^{2n+m_2} + 2^{n+m_1}(n + m_2)$. For example, for $m = 5n$, setting $m_1 = 3n$ and $m_2 = 2n$ we get complexity at most $2^{4n}(4n)$ less than the 2^{5n} required for exhaustive search.

Thus, V can be considered as an entropy buffer auxiliary to the computation of the final output(s).

5.5 Effect of Salt

The persistent cryptovariable *shash* is designed to transition the non-degenerate (through primary computation) salt dependency into successive computations through PRF keying. This ensures greater salt dependency for the final cryptovariable.

Therefore, since $S = \{0,1\}^k$ $k \geq 128$, we expect that the space-time complexity for complete or partial precomputation should be out of reach of even the most well resourced of adversaries.

On the other hand, we observe that high dependency on *salt* allows for multiple independent instances of $F_{N,M}$. This allows $F_{N,M}$ to be used for key derivation for multiple independent applications, settings, sessions etc. even with minimal rekeying [4, 5].

5.6 Various Observations

In this section we discuss various elements of the design of $F_{N,M}$ and their impact on security.

We start by discussing the effect of variably rekeying PRF as follows. Since $HMAC_p = HMAC_{k\,=\,H_{IV}(p)}$ for all $p \in PW$ such that $|p| > block\,length$, the entropy of the source is distilled into the n-bit cryptovariable, k. Therefore, if all instances of PRF were based on $HMAC_p$, the attack complexity for a variant of the meet-in-the-middle attack in Section 5.3 for all sources PW with string lengths beyond block length and $\mathcal{H}(PW) > 2n$ would be below exhaustive search.

The use of counter mode in PRF calls of *line 8* and *line 14* forces different inputs into each successive call. This reduces the incidence of collision induced cycles essential for the short cut attacks in the proofs of [6] and [13]. Therefore, it ensures $F_{N,M}$ minimally suffers from effects of reuse of internal values.

The requirement for random access to X in *line 14* of the algorithm imposes memory constraints on $F_{N,M}$-computing circuits. Thus, it impacts on the cost of special purpose key search circuits while ensuring efficiency of computation on general purpose computers [6].

Finally, we note that the inclusion of iteration count c and salt s in *lines 8, 11* and *17 (variously)* should allow us to make a stronger claim of security than set out in Section 3. Indeed, assuming a random H, we expect the $F_{N,M}(p, s_i, c_i, n)$ queries for $0 \leq i \leq q$ (for some threshold value q and pairs (s_i, c_i)) to portray some random behaviour.

In particular, allowing for the argument of degeneracy of the salt effect [5], we expect at the bare minimum the value of *shash* in *line 11* to not only be pseudorandom but dependent on both s (directly) and c (via digest) - an argument extensible to values in V (via the PRF key and elements of X). A similar argument can be made for the final cryptovariable (via PRF key *shash* from *line 17*).

Therefore, we believe the adversary can gain little traction by making supplementary queries to the oracle, $F_{N,M}(p_0, (.), (.), n)$.

6 Efficiency analysis

6.1 Software Implementations

The feedback mode in the time critical PRF calls of *line 8* and *line 14* impose dependency constraints on parallel implementations. Therefore, efficient implementation on all platforms including those with modern features such as multicore CPUs is dependent on the state of the art in PRF implementation.

In particular, while multiple calls to HMAC_k for fixed k can be optimized by precomputing [11]

$$\begin{aligned} k_0 &= H_{IV}(k^+ \oplus ipad) \\ k_1 &= H_{IV}(k^+ \oplus opad) \end{aligned}$$

(a computation with inherent parallelism), subsequent and time critical calls

$$HMAC_k(data) = H_{k_1}(H_{k_0}(data))$$

retain high levels of dependency (based on both the structure of *data* in both *lines* and the design of Merkle-Damgard hash functions).

On the other hand, computing the moduli in *line 13* and *line 15* each require one clock cycle for all values of N and M that are powers of 2. Therefore, independent of the compression function for H, there is little room for software optimisation.

For completion, an example implementation (e.g. the reference implementation in the submission package) on a 1.6 GHZ Intel Core 2 Duo Processor running the GCC compiler can compute M3lcrypt_{SHA256} (using the minimum parameters for the algorithm) in 258 milliseconds which translates to 3.876 evaluations of the PBKDF per second. In comparison, at creation in 1977, *crypt* could be evaluated about 3.6 times per second on a VAX-11/780 [8].

6.2 Hardware Implementations

The availability of large random access memory (RAM) in software implementations shift the implementation bottleneck from random access memory (RAM) to optimal implementation of the PRF.

On the contrary, we can assume that efficient hardware for the hash function exists (e.g. for standardised functions such as the SHA2 family). Possibilities for further customisation (e.g. external pipelining and/or other extensive parallelism) are contingent on the availability and cost of RAM.

For example, there exist a myriad of time/memory trade-offs scaling from the entire RAM requirement for X to any fraction of this requirement. However, since X has [pseudo]random, independent and unpredictable values [for any polynomial time algorithm] any further non-trivial time/memory trade-off increases the number of auxiliary computations required to process "random" values $X[k]$ for *line 14* of the algorithm.

In particular, any values $X[k]$ (for some $k = \mathit{digest} \,\%\, N$) at *line 13* not in memory will either have to be computed from scratch or from some point further down (in RAM) the computation chain.

Therefore, assuming large memory requirement for X, massively parallel key search machines may be [area-time] costly.

7 Conclusion

We have described a new password based key derivation function, M3lcrypt (canonical M3lcrypt$_H$) which is secure from adversaries with oracle access to PRF (the pseudorandom function family based on H). Further, we conjecture security from adversaries with oracle access to both PRF and the function instance itself (i.e. parameter modifying adversaries).

In practice, however, the key derivation function, though essential, may not eliminate the effect of χ_u. User choice and management of passwords (along with other cryptographic key material) remain by far the greatest security threat in password based cryptography. Therefore, password policies (rules that define the organisational χ_u) play a critical role in the provision of secure password based services.

References

[1] M. Bellare, R. Canetti and H. Krawczyk, *Keyed Hash Functions for Message Authentication*, Advances in Cryptology Crypto 96, Springer-Verlag, 1996.

[2] C. Percival and S. Josefsson, *The Scrypt Password-Based Key Derivation Function*, IETF Internet Draft, 2012.

[3] S. Goldwasser and M. Bellare, *Lecture Notes on Cryptography*, July 2008.

[4] H. Krawczyk, *Cryptographic Extraction and Key Derivation: The HKDF Scheme*, Crypto'2010, LNCS 6223, 2010.

[5] B. Kaliski, *PKCS #5: Password-Based Cryptography Specification Version 2.0*, RFC 2898, 2000.

[6] J. Kelsey, B. Schneier, C. Hall and D. Wagner, *Secure Applications of Low-Entropy Keys*, Proceedings of the First International Workshop ISW 97, Springer-Verlag, 1998.

[7] D. Klein. *Foiling the Cracker: A Survey of and Improvements to Password Security*, Proceedings, UNIX Security Workshop II, August 1990.

[8] N. Provos and D. Mazieres. *A Future-Adaptable Password Scheme*, USENIX Annual Technical Conference, USENIX 99, The Advanced Computing Systems Association, 1999.

[9] J. Killian and P. Rogaway, *How To Protect DES Against Exhaustive Key Search Attacks*, Advances in Cryptology - CRYPTO 96, Springer-Verlag, 1996.

[10] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Wiley & Sons, Second Edition, 1996.

[11] W. Stallings. *Cryptography and Network Security: Principles and Practice*, Prentice Hall, Second Edition, 1998.

[12] D.R. Stinson, *Cryptography: Theory and Practice*, Second Edition, Chapman & Hall, 2002.

[13] F.F. Yao and Y.L. Yin, *Design and Analysis of Password-Based Key Derivation Functions*, CT-RSA 2005.