New HMAC Message Patches: Secret Patch and CrOw Patch

Student Name: Nishant Sharma

IIIT-D-MTech-CS-IS-12-013
May 2, 2014

Indraprastha Institute of Information Technology
New Delhi

Thesis Committee
Dr. Somitra Sanadhya, IIIT Delhi (Chair)
Dr. Shweta Agrawal, IIT Delhi
Dr. Debajyoti Bera, IIIT Delhi

Submitted in partial fulfilment of the requirements
for the Degree of M.Tech. in Computer Science,
with specialization in Information Security

Keywords: HMAC, related key attack, colliding key pairs, indifferentiability, distinguisher, internal state recovery

Certificate

This is to certify that the thesis titled **"New HMAC Message Patches: Secret Patch and CrOw Patch"** submitted by **Nishant Sharma** for the partial fulfilment of the requirements for the degree of *Master of Technology* in *Computer Science & Engineering (Information Security)* is a record of the bonafide work carried out by him under our guidance and supervision in the Security and Privacy group at Indraprastha Institute of Information Technology, Delhi. This work has not been submitted anywhere else for the reward of any other degree.

Dr. Donghoon Chang
IIIT Delhi

Dr. Somitra Sanadhya
IIIT Delhi

Abstract

HMAC or keyed-hash message authentication code is a security implementation using cryptographic hash function (where hash function is iterative i.e. classical Merkle-Damgård construction [7] [12]) and a secret key. It was designed by Bellare, Canetti and Krawczyk in 1996 [3]. It was subsequently adopted by IETF working group as RFC 2104 [10] and made a standard for authentication in secure internet protocols. It is widely used in banking industry and secure web connections via its use in TLS and IPSEC. The security of HMAC was proven in [2] but this proof of security does not consider related key model.

In Asiacrypt 2012, Peyrin *et al.* [13] showed related key attacks against HMAC design. Following this, they also proposed a patching scheme for standard HMAC and claimed that the proposed patch thwarts their attacks. However they didn't provide any security proof/explanation for the same.

In this work, we show that the patch proposed by Peyrin *et al.* [13] will not disallow their attack for the HMAC construction for certain hash functions. We emphasize that our approach is valid for the general HMAC construction and not for the standardized version of HMAC, which uses a specific hash function, namely SHA-1. We show that the related key attacks of Peyrin et al. still work when HMAC is constructed from a "good" cryptographic hash function satisfying collision resistance, preimage resistance and second preimage resistance under certain circumstances.

On similar lines, in Crypto 2012, Dodis *et al.* [8] showed differentiability attacks on HMAC based on weak keys (ambiguous and colliding). In order to thwart the two types of attacks, we propose two tweaks for thwarting the both attacks. One of them requires using wrapper patch, while the other uses a new padding scheme for HMAC. Our first modification requires our new patching schemes for HMAC which ensure the safety of HMAC scheme from the attacks discussed by Peyrin *et al.* [13]. Our second modification ensures that the HMAC will not have any colliding keys hence thwarting the attack of Dodis *et al.* [8]. Thus we show that the HMAC with one of our patches and new padding scheme is safe from cycle detection based related key attacks discussed by Peyrin *et al.* [13] and indifferentiability attacks using colliding pairs by Dodis *et al.* [8].

Acknowledgments

I dedicate this thesis work to my grandparents and parents. Thank you for being so supportive, without your support this would not have been possible.

I would like to thank Dr. Somitra Kumar Sanadhya and Dr. Donghoon Chang. They introduced me to the area of cryptography and without them it would have been impossible to do this work in one year. Dr. Somitra's dedication and devotion to his work is incomparable. I will always remember the nights that sir spent in his office for helping me. Dr. Donghoon always motivated me to work on trending problems in cryptography and also advised me about my career. I was really fortunate to work with such people.

I thank Dr. Pankaj Jalote and all founding members of IIIT Delhi for creating such a wonderful institute and giving me an opportunity to study here.

I would also like to take this opportunity to thank everybody who have been a been part of my life at IIIT-D, especially Sandipan Biswas, Rohit Romley, Rohit Jain, Gajendra Waghmare, Prateek Sharma, Aritra Dhar, Naufal, Veeru and many people with whom I had fun and learned many things.

Last but not the least, I thank Dr. Shweta Agrawal from IIT-Delhi for accepting to be a part of my thesis committee as the external examiner.

Nishant Sharma,
MT12013

Contents

List of Figures

List of Tables

Chapter 1

Introduction

From the beginning of human race, correct information has power to transform the world. Advancement of society increased the importance of correct information drastically which also led to the need of information protection. As a result mathematicians/scientists defined some complex methods of keeping information secure in a place or in transit (communication) and termed this area as Cryptography. The word cryptography is taken from Greek κρυπτός (means "hidden" or "secret"); and γράφειν, graphein (means "writing") or -λογία, -logia (means "study"), respectively. Cryptography is an art of hidden writing or study of methods used for hiding secret information. Cryptography is helping people to protect information from ancient times to present era of internet. As time progressed, cryptographic techniques as well as area of applications also changed. Cryptographers and cryptanalyst's war, which is going on from centuries showed world transition from primitive ceaser cipher to sophisticated AES. Today, cryptography is almost everywhere as the last defense against any kind of attack, let it be online transactions on HTTPS/TLS or file encryption or password storage or secure ID tokens. Cryptography is used to provide confidentiality, authenticity and integrity.

1.0.1 Ensuring confidentiality

Suppose we have two friends Alice (A) and Bob (B) who like each other and want to exchange their messages over unsecured channel e.g. B writes his message on a paper and throws it towards A's window. Now, problem is that A's dad, Charlie (C) doesn't like this and if he finds out what kind of messages are exchanged, he may thrash A. Here, they need a strategy so that even if (C) catches a message, he can't figure out what is the information contained in it. For that, (A) and (B) must have a secret element, this element may be (i) scheme, also known as the encryption scheme or (ii) secret input, also known as the key. So, A and B can either use a secret scheme or publicly known scheme with secret key to communicate secretly. The latter model is preferred in present scenario due to its ease of

use. This mechanism of protecting secret communication is known as Encryption. Encryption used in today's scenario for protecting electronic information. It can be broadly classified into two types based on key types:

- **Symmetric encryption:** If both parties i.e. A and B use the same key, then they are said to be using symmetric encryption. This key is exchanged by some mechanism before starting communication by using some secure channel. Some popular symmetric encryption algorithms are DES, AES, 3DES, Blowfish.
- **Asymmetric encryption:** If both parties i.e. A and B are using different keys, then they are said to be using asymmetric encryption. In this scenario every party has a pair of keys consisting of public key and private key. Public key is shared with public but private key is kept secret. In case a party say A wants to send a message to other party say B, then A will encrypt the message with B's public key and send it. Now, the keys are mathematically created in such a way that only the private key can decrypt the cipher encrypted with the corresponding public key. So, B will receive the message and then decrypt it using his key. Some popular asymmetric encryption algorithms are RSA, Diffie Hellman Key Exchange, AlGamal.

The problem of confidentiality can be handled by using encryption techniques. But it still can't ensure the integrity and authenticity of message.

1.0.2 Ensuring Integrity and Authenticity

Ensuring integrity of a message means that the message which is sent by the sender reaches the receiver as it is i.e. no third party or communication errors or data loss can change the content without getting noticed in receiver side. A simple hash function H can be used to provide integrity check. A sends hash tag t of message along with the message to the receiver so that he can recompute the hash tag t, match it and can ensure that the message is not altered. But if adversary C from same scenario intercepts the message, he can change the message and generate a valid tag t using same algorithm H. This happens because in algorithm H used for hash tag generation is public. As a countermeasure, A instead of using simple Hash, can use keyed Message Authentication Codes (MACs) which also require a secret key k to generate the hash tag. Now, even if C intercepts the message he can change the message but as he doesn't know the key k, he can't generate a valid tag for the message. Ensuring authenticity of a message means that the receiver can somehow ensure that the message is sent by the legitimate sender. In keyed MACs the key k is only known to the sender and the receiver, hence no one else can

generate a valid MAC for message, thus ensuring the authenticity. The problem of integrity and authenticity can be handled by using keyed MACs.

1.1 Cryptographic Hash Function

Cryptographic hash functions are used as building blocks in many cryptographic protocols and applications. A hash function can take arbitrary size input and produce a finite size output. A hash function construction consist of two main parts:

- **Compression function:** A compression function is an underlying function in a Hash construction which take finite size input and produces finite size output.
- **Iterator:** An iterator setup repeats the compression function so that arbitrary size input can be accepted.

1.1.1 Construction of Hash Function

A cryptographic hash function can be constructed using a block cipher or some special structure with collision resistant compression function. We will only consider the latter here. The Merkle-Damgård Construction is famous for construction hash functions.

Merkle Damgard Construction

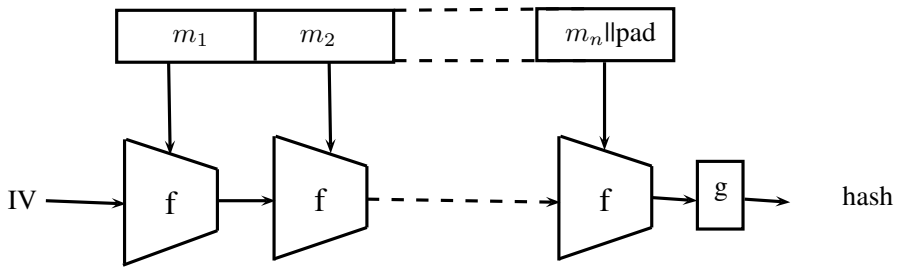

Figure 1.1: Merkle Damgard Construction

The classical Merkle-Damgård construction (MD) [7] [12] was described in Ralph Merkle's Ph.D. thesis in 1979. Ralph Merkle [12] and Ivan Damgård [7] independently proved that the structure is sound: that is, if an appropriate padding scheme is used and the compression function is collision-resistant, then the hash function will also be collision resistant. As the compression function can only accept finite size output, we need to pad the arbitrary size input into multiples of input size it accepts. This is done by padding the last message with appropriate number of bits.

The MD construction is shown in Figure 1.1. The f here is underlying compression function and g is the finalization function.

Some popular Hash functions based on MD construction are MD5 [14] and SHA-1 [1].

1.1.2 Good Hash Function

A good hash function is a hash function wich satisfy the following conditions:

Preimage Resistance

If there exists no adversary A which can find x given h such that $h = H(x)$ for hash function H. Then, hash function H is said to be preimage resistant.

Second Preimage Resistance

If there exists no adversary A which can find pair x,x' such that $H(x) = H(x')$ but $x \neq x'$ for hash function H. Then, hash function H is said to be second preimage resistant.

Collision Resistance

If there exists no adversary A which can find pair x,x' such that $H(x) = H(x')$ for hash function H. Then, hash function H is said to be collision resistant.

In [6], Coron *et al.* suggested that a hash function should behave like a random oracle. In our work we also show that how even a good hash function fails if it is not random.

1.2 HMAC

HMAC or keyed-hash message authentication code is a security implementation using cryptographic hash function (where hash function is iterative i.e. MD construction and a secret key. It was designed by Bellare, Canetti and Krawczyk in 1996 [3]. It was subsequently adopted by IETF working group as RFC 2104 [10] and made a standard for authentication in secure internet protocols. It is widely used in banking industry and secure web connections via its use in TLS and IPSEC. The security of HMAC was proven in [2] but this proof of security does not consider related key model.

The HMAC construction [3], using a hash function H, is defined as HMAC(K,M) = $H(K \oplus opad||H(K \oplus ipad||M))$, where `ipad` and `opad` are constants defined as `ipad` = `0x363636… 36` and `opad` = `0x5C5C5C… 5C`. H is any cryptographically secure hash function which takes an arbitrary sized input message M and produces a finite n bit output h after finalization of l bit internal state. K is the message authentication secret key shared between the two communicating parties. Let d be the block length of HMAC which depends on the choice of underlying hash function H. Let $\overline{K}$ be the padded version of the key K i.e $\overline{K} = K \parallel 0000...$, where $|\overline{K}| = d$. We will only consider key K of length d or $d-1$ so we will consider $\overline{K} = K$ for the rest of the thesis. The HMAC or HMAC-H(K, M) construction is explained in Figure 1.2.

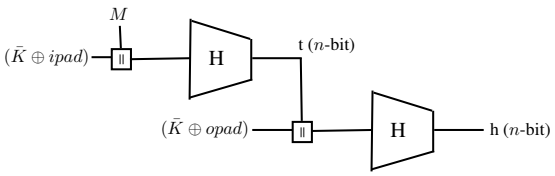

Figure 1.2: The HMAC construction

The security of HMAC is proven in terms of its unforgeability. The HMAC construction as shown in Figure 1.2, uses the same key K to produce two keys: $\overline{K} \oplus ipad$ and $\overline{K} \oplus opad$. The relationship between these two keys makes this construction prone to related key attacks.

Related key attacks are applicable when some relation between the two keys is known, although the keys themselves may not be known. These attacks have serious impact. In [13], the authors showed how this relationship can lead to Distinguishing-R, Distinguishing-H, internal state recovery and forgery attacks on HMAC (we describe the Distinguishing-R attack later in this work and make some comments about the others). The authors then proposed a patch for HMAC to prevent the related key attacks. However they have not provided any proof of security or explanation of the patched HMAC against related key attacks. On similar lines, Dodis *et al.* [8] showed differentiability attacks against HMAC using colliding key pairs.

In this work we explain how the patch proposed by Peyrin *et al.* [13] doesn't work for the general HMAC construction. Our contribution is to provide an efficient patch which prevents related key attacks based on cycle detection described in [13] on standardized HMAC as well as the general HMAC construction. We also propose a modification in the padding scheme of HMAC to ensure that the colliding keys based attack of Dodis *et al.* [8] no longer works. We also consider new generic attacks on Hash based MACs discussed in [9]. Their approach is to attack underlying hash function rather than exploiting the HMAC structure so, is not related to our work.

Chapter 2

Related work

We will now briefly desrcibe the work done by Peyrin *et al.* and Dodis *et al.*. We will try to understand their approach and then try to fix the problems stated.

2.1 Peyrin *et al.*'s work

Distinguishing-R, distinguishing-H, internal state recovery and forgery attacks against HMAC scheme were shown in [13]. The HMAC is based on balanced hash functions as dictated by Mihir Bellare *et al.* in [4]. In this chapter, we briefly describe the crucial elements which are common to all these attacks.

The HMAC design is inspired by NMAC [2]. However, there is a crucial difference between NMAC and HMAC. While the NMAC uses two independent keys K_{in} and K_{out}, HMAC uses a single key K and uses the XOR of this single key with two predefined constants `ipad` and `opad` to generate two values K_{in} and K_{out}, which are used as secret keys. The $H_{K_{in}}(M) = H(K \oplus ipad||M)$ and $H_{K_{out}}(M) = H(K \oplus opad||M)$ are the inner and outer hash functions of HMAC with inner and outer keys K_{in} and K_{out} respectively. The attack of [13] utilizes the relationship between these two keys to construct attacks against HMAC. For comparison, the two designs are defined below. Note that both the designs use an underlying hash function H. To denote calls to the same hash function with different IV values, we use the notation H_{IV}.

$$\begin{aligned} \text{NMAC}(K_{in}, K_{out}, M) &= H_{K_{out}}(H_{K_{in}}(M)), \\ \text{HMAC}(\overline{K}, M) &= H_{\overline{K} \oplus \texttt{opad}}(H_{\overline{K} \oplus \texttt{ipad}}(M)). \end{aligned}$$

The attack in the related key scenario is based on the known relation between the two keys K and K', which themselves are not known to the attacker. Treating HMAC as a special case of NMAC, we can see that K_{in} and K_{out} are derived from

the same key as $K_{in} = K \oplus \texttt{ipad}$ and $K_{out} = K \oplus \texttt{opad}$, where K is the secret key and ipad and opad are publicly known constants.

Now consider a related key scenario in which the keys for two invocations of HMAC are K and $K' = K \oplus \texttt{ipad} \oplus \texttt{opad}$. By the choice of the keys, it is evident that the two invocations of HMAC use different order of internal functions $H_{K_{in}}$ and $H_{K_{out}}$. That is,

$$\begin{aligned}
(i) \quad \text{HMAC-H}(K, M) &= H_{K_{out}}(H_{K_{in}}(M)), \text{ and} \\
(ii)\ \text{HMAC-H}(K', M) &= H_{K' \oplus \texttt{opad}}(H_{K' \oplus \texttt{ipad}}(M)) \\
&= H_{K \oplus \texttt{ipad}}(H_{K \oplus \texttt{opad}}(M)) \\
&= H_{K_{in}}(H_{K_{out}}(M)).
\end{aligned}$$

Figure 2.1 describes the notations used in subsequent chapters. It also shows HMAC-H(K, M) and HMAC-H(K', M) in terms of inner and outer hash functions.

Figure 2.1: Explanation for basic functions notations used in subsequent chapters

In [13], the authors showed four different types of attacks on HMAC, namely, Distinguishing-R, Internal State Recovery, Distinguishing-H and forgery attack.

Distinguishing-R Attack

Let the keys K and K' be related by a known expression. Given two oracles F_K and $F_{K'}$ instantiated with HMAC-H(K, M) and HMAC-H(K', M) in the first case and with random functions R(K, M) and R(K', M) in the second case, if an adversary $\mathcal{A}$ can distinguish between these two cases with non-negligible probability then $\mathcal{A}$ is said to successfully perform distinguishing-R attack on the HMAC scheme.

The advantage of the attacker is given by

$$Adv(\mathcal{A}) = |\Pr[\mathcal{A}^{HMAC(K,M),HMAC(K',M)} = 1] - \Pr[\mathcal{A}^{R(K,M),R(K',M)} = 1]|.$$

Distinguishing-H Attack

Let the keys K and K' be related by a known expression. Given two HMAC oracles $HMAC(K, M)$ and $HMAC(K', M)$ instantiated with known underlying hash h in the first case and with random underlying function r in the second case, if an adversary $\mathcal{A}$ can distinguish between these two cases with non-negligible probability then $\mathcal{A}$

is said to successfully perform distinguishing-R attack on the HMAC scheme.

The advantage of the attacker is given by

$$Adv(\mathcal{A}) = |\Pr[\mathcal{A}^{HMAC^h(K,M),HMAC^h(K',M)} = 1] - \Pr[\mathcal{A}^{HMAC^r(K,M),HMAC^r(K',M)} = 1]|.$$

Internal State Reovery

Internal State Recovery is said to be done successfully if attacker can recover the l-bit internal state of underlying hash H from n-bit output. This ability is used to mount Distinguishing-H and Forgery attacks.

Forgery

Forgery is said to be done successfully when attacker can generate a valid tag t for a message m, which it never queried to HMAC.

- If the message is chosen by attacker then it is Existential Forgery.
- If the message is chosen by challenger the it is Universal Forgery.

We discuss the Distinguishing-R attack below, and leave the rest as all other attacks as they are also based on the same idea. Please refer [13] for details of these attacks.

Let us consider two oracles instantiated with HMAC-H(K, M) and HMAC-H(K', M). To proceed with the attack, we generate paths for the corresponding oracle computations and detect cycles in the path.

Path refers to sequence of outputs of specific functions when the previous output is used as input for the next step. A path may have a cycle so for better reference, we divide path into two parts, called the ``Cycle part" (the part comprising a cycle) and a "Tail part" (the part before entering into the cycle).

Cycle refers to a sequence which if encountered in path, repeats after a fixed interval for infinite time. This interval is dubbed as cycle length.

Cycle detection attack

refers to attack on cryptographic oracle where adversary obtains a path from oracle and then tries to find a cycle in the path. By using this attack, the adversary can launch a distinguishing attack as the probability for the existence of a cycle may be very small for a random oracle.

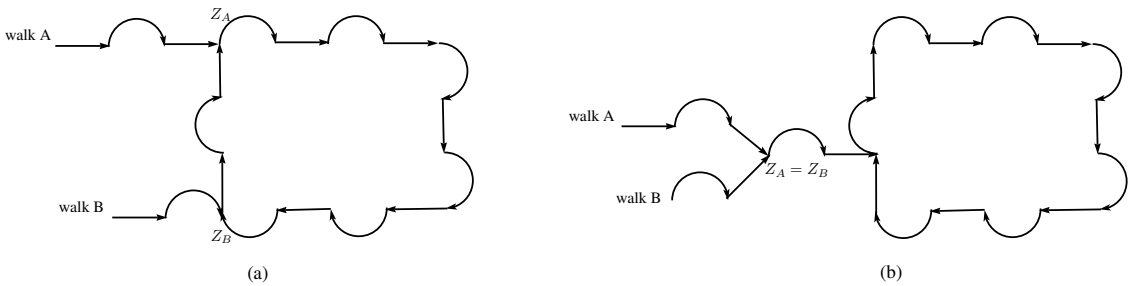

Figure 2.2: (a) Walk A and B entering into the same cycle at point Z_A and Z_B respectively. (b) Walk A and B entering into the same cycle in synchronization i.e. $Z_A = Z_B$.

Let $y = f(K, x)$. We define the oracle $\mathcal{O}_i^f(K, M)$ as repeatedly calling $f(K, x)$ for i times with the starting value $x = M$. That is, $\mathcal{O}_i^f(K, M) = f(K, \mathcal{O}_{i-1}^f(K, M))$. For example, a 2 step oracle call can be expressed as $\mathcal{O}_2^f(K, M) = f(K, f(K, M))$. A collision refers to a match in the oracle outputs at different steps, i.e. $\mathcal{O}_i^f(K, M) = \mathcal{O}_j^f(K, M)$ where $i \neq j$.

The attack can be executed in three steps as follows:

1. Walk A: Choose a random n-bit string and generate a path using oracle HMAC-H(K, M) for $2^{n/2} + 2^{n/2-1}$ steps till collision happens. If no collision is found or if the collision happens in the first $2^{n/2}$ queries, output 0.

2. Walk B: Choose a random n-bit string and generate a path using oracle HMAC-H(K', M) in the same manner.

3. If the length of the cycles in walk A and walk B are equal, then output 1 otherwise output 0.

The distinguisher works because the probability of collision in the two walks for a random oracle is significantly different from the HMAC oracle. Thus, the distinguishing-R attack can be easily executed by comparing the cycle lengths. The other three attacks described in [13], namely internal state recovery, forgery and distinguishing-H, also work on the same principle.

A brief explanation of why the attack works is as follows. If cycle generated in walk A and walk B have the same length then it is the same cycle. In normal scenario, the entry points of walk A and walk B (depicted as Z_A and Z_B in Fig. 2.2(a)) are at different positions. We know that in a cycle, the input and the output of HMAC-H(K, M) will be the intermediate internal state of HMAC-H(K', M) and vice versa. However they can't be figured out due to the different entry points. If we have a synchronized cycle, i.e. collisions taking place on the tails of Path A and Path B so that they enter the cycle in synchronization (i.e. $Z_A = Z_B$ as shown in Fig. 2.2(b)), then the internal state can be recovered and the attacker can perform all these attacks. For details, we direct the reader to [13].

The main idea behind all these attacks is to obtain a cycle or a synchronized cycle. If we can prevent the adversary from obtaining a cycle then all these attacks will be thwarted.

2.1.1 Patch proposed by Peyrin *et al.*

In [13], the authors state that the choice of $ipad$ and $opad$ is not anecdotal. If any other random pair of constants is used as $ipad$ and $opad$ then the attacks may work for all key sizes. Peyrin et al. show that related key pairs which allow disntinguishing attacks only exist for keys of length $\geq d - 1$, where d is the block length of the underlying hash function.

The authors then propose few patching schemes which can thwart their attacks by preventing the formation of cycles. The schemes proposed and some comment on the proposal as provided in [13] are as follows:

1. Use of different IVs in inner and outer instances of HMAC. It was rejected as it requires modification of the HMAC implementation.
2. Truncating the output of HMAC. It was also rejected as the expected generic security of MAC algorithm reduces due to this change.
3. XORing some distinct constants to inner and/or outer hash calls. As explained in [13], this patch does not work since the attacker can suitably modify its query strategy and can still get synchronized chains.
4. Adding an extra bit to the input of outer hash call. It was also rejected since the attacker can still get synchronized chains by modifying his query strategy.
5. To prepend a 0 bit (or byte) to the input message before passing it to the HMAC construction. The authors of [13] claim that this patch successfully prevents adversary from getting cycles and the additional overhead of ading 1 bit (or byte) to the message is insignificant. This patch has the additional advantage of the feasibility of being implemented by means of a message wrapper while keeping the HMAC implementation unchanged.

2.1.2 Patch $P0$

As mentioned above, after comparing all the proposals for the different types of patches, the authors of [13] proposed the patching scheme of prepending a 0 bit (byte) to the input message before feeding it to HMAC as their final patch. For our analysis, we will refer to this patch as the patch $P0$. The patch $P0$ is defined as

$$P0(M) = 0||M$$

where M is message of any length and resulting HMAC^{P0}-H(K, M) (also denoted as HMAC- H$(pad^{0^*}(K), P0(M))$) is defined as

$$\text{HMAC}^{P0}\text{-H}(K, M) = \text{HMAC-H}(K, P0(M)).$$

$$= \text{HMAC-H}(K, 0 \parallel M)$$

This construction is defined in Fig. 2.3.

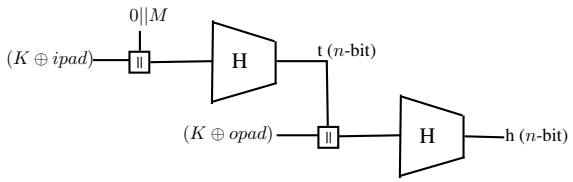

Figure 2.3: HMAC^{P0}-H(K, M) construction.

2.2 Dodis *et al.*'s work

In [8], the authors described two kinds of weak keys (i.e. ambiguous and colliding keys) which when used with HMAC-H(K, M) can allow an attacker to mount attacks on the scheme. The authors suggested that the only way to avoid such attacks is not to use these weak keys. These pairs can be avoided by using keys of fixed length $|K| < d - 1$. Using these weak keys authors in [8] showed differentiability attacks on HMAC-H(K, M). So we will analyse our HMAC^{SP_K}-H(K, M) for weak key pairs. For better understanding we will first discuss the padding schemes.

2.2.1 Key padding

In HMAC scheme the key size $|K|$ should be equal to block size d of underlying hash function H. So if $|K| < d$ then we have to pad the key with some bits to make it equal to d. For this purpose key padding schemes are used. However if $|K| = d$ then no padding is done.

Pad$^{0^*}(x)$ padding scheme

or 0^* padding, defined as $pad^{0^*}(x) = x||0^v$ where $v = d - |x|$.

2.2.2 Ambiguous keys

are those key pairs K_1 and K_2 such that $K_1 \neq K_2$ but $pad^{0^*}(K_2) = pad^{0^*}(K_1) \oplus ipad \oplus opad$. As a result $f_{K \oplus ipad} = f_{K' \oplus opad}$ and $f_{K \oplus opad} = f_{K' \oplus ipad}$. These are also known as related keys.

2.2.3 Colliding keys

are those key pairs K_1 and K_2 such that $K_1 \neq K_2$ but $pad^{0^*}(K_1) = pad^{0^*}(K_2)$ due to which HMAC(K_1,M) = HMAC(K_2,M).

Chapter 3

Our Contribution

3.1 Motivation and Research Problem

We discussed in earlier sections that the HMAC is widely deployed in industry, Hence any attack on HMAC can't be ignored. The Peyrin *et al.* [13] showed some attacks which were demonstrated earlier with higher complexities in single key scenario but they showed that these attacks can be done on a lower complexity in related key model. They also provide patch for these attacks but no explanation/proof is provided. Whereas Dodis *et al.* [8] showed that ambiguous pairs may lead to indiffrentiability attacks. So, we decided to work on these problems and try to fix them.

Research Aim:

1. Study and verify Peyrin *et al.*'s patch.
2. **Provide efficient patch:** If Peyrin *et al.*'s patch is not secure the only defense to prevent these attacks will be to increase the internal state of implementation. This measure with increase the effort for adversary but it will also affect the performance of HMAC. Secondly, the HMAC is widely deployed and it is not easy to replace older implementation with new implementation (with larger internal state) from everywhere. Hence we must find some efficient and easy way to patch (so that we need not to change the whole implementation) this problem.
3. Prevent indifferentiability attacks on HMAC.

3.2 Insecurity of patch proposed by Peyrin *et al.*

Note that no proof of security or explanation of the patch is provided in [13]. We analyse the construction HMAC^{P0}-H$^{P0}(K, M)$ by checking it against various modifications and assumptions. To analyse, we may modify underlying primitives but not the fundamental HMAC construction. We emphasise that we allow the attacker to tamper the output of oracle HMAC-H (or HMAC-H') before the next call to the same or a different oracle. However, the attacker can't temper within the HMAC construction (he can only tamper between two calls to the HMAC oracle). In Fig. 3.1 we have depicted the attack/modification area. By introducing changes at the place marked as ``attack" in this figure, we analyse the patch $P0$.

Collision Finding Adversary

An adversary $A(H, \epsilon, t)$ is known as a collision finding adversary if it can find collisions in the hash function H in time t with advantage ϵ. ϵ is defined as

$$\epsilon = Pr[x, x' \leftarrow A : H(x) = H(x') \,\&\, x \neq x'].$$

Preimage Finding Adversary

An adversary $A(H, \epsilon, t)$ is known as a preimage finding adversary if it can find preimage in hash function H in time t with advantage ϵ. ϵ is defined as

$$\epsilon = Pr[x \leftarrow A : h = H(x)].$$

Second Preimage Finding Adversary

An adversary $A(H, \epsilon, t)$ is known as a second preimage finding adversary if it can find second preimage in hash function H in time t with advantage ϵ. ϵ is defined as

$$\epsilon = Pr[x' \leftarrow A : H(x) = H(x') \& x \neq x'].$$

3.3 Hash H is Collision, Preimage and Second Preimage Resistant

The security of HMAC-H(K, M) is depicted in terms of unforgeability but depends on the underlying hash function H. We analyze the security of HMAC^{P0}-H(K, M)

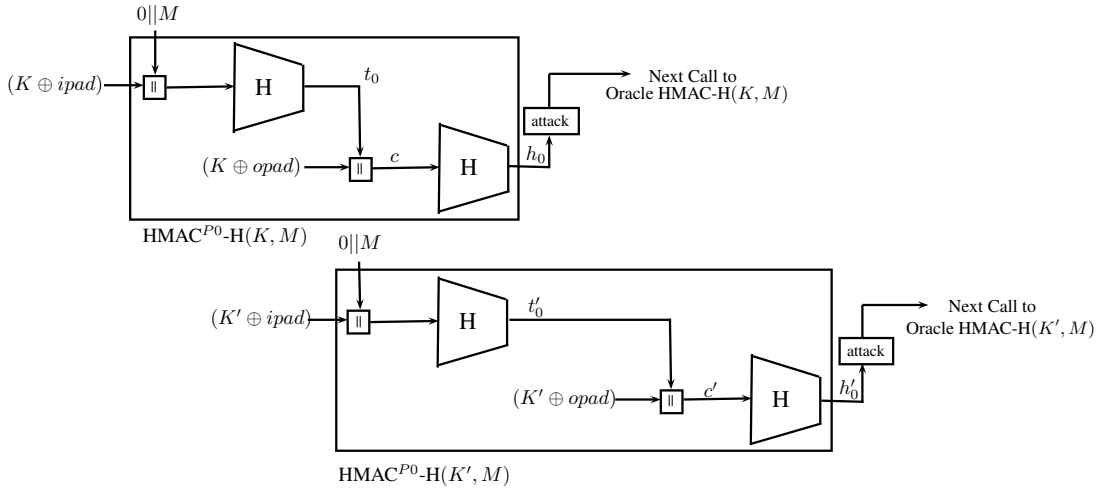

Figure 3.1: HMAC^{P0}-H(K, M) and HMAC^{P0}-H(K', M) behave like a black box, hence an attacker can only modify the sequence between two consecutive calls to the oracles.

with the assumption that the underlying H is a ``good" hash function i.e. H is collision resistant, preimage resistant and second preimage resistant.

Hash construction H^{P0}

Let H^{P0} be a collision resistant (CR), preimage resistant (PR) and 2nd preimage resistant (Second PR) hash function, which uses a CR, PR and Second PR hash function H as its underlying function and prepends 0 to its output. Patch $P0$ is defined as

$$P0(M) = 0||M$$

where M is a message of arbitrary length. H^{P0} is defined as

$$H^{P0}(M) = P0(H(M)) = 0||H(M) = 0||h.$$

The construction H^{P0} is described in Fig. 3.2.

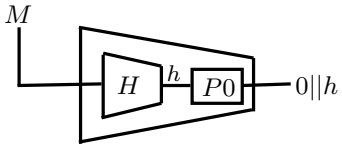

Figure 3.2: Hash function H^{P0} which is collision, Preimage and 2nd Preimage resistant. The block $P0$ takes an input h and outputs h prepended with 0.

H^{P0} is CR, PR and Second PR

We discussed earlier that the security of HMAC-H(K, M) relies on the underlying hash function. We show that the construction H^{P0} satisfies the essential properties of a good hash function, it is collision resistant (CR), preimage resistant (PR) and second preimage resistance (Second PR).

Theorem 1. *If there exists a collision finding adversary $A(H^{P0}, \epsilon, t)$ then there also exists a collision finding adversary $B_A(H, \epsilon, t')$.*

Proof: Let us consider an efficient adversary $A(H^{P0}, \epsilon, t)$ which can find collisions for H^{P0} with more than negligible probability.

Algorithm 1 Collision finding adversary B_A for H

INPUT: A hash function H

OUTPUT: x_1,x_2 where $H(x_1) = H(x_2)$ but $x_1 \neq x_2$

x_1,$x_2 = A(H^{P0}, \epsilon, t)$

return x_1,x_2

To find a colliding pair for H, we require an efficient collision finding adversary $A(H^{P0}, \epsilon, t)$ where time $t = t'$ + time consumed for calculating $P0$ and t' is the time consumed by adversary B_A. But hash function H is collision resistant, so existence of such an adversary B_A is not possible.
$\Rightarrow$ H^{P0} is collision resistant.

Theorem 2. *If there exists a preimage finding adversary $A(H^{P0}, \epsilon, t)$ then there also exists a preimage finding adversary $B_A(H, \epsilon, t')$.*

Proof: Let there be an efficient adversary $A(H^{P0}, \epsilon, t)$ which when provided with h, can find x such that

$$h = H^{P0}(x).$$

Algorithm 2 Preimage finding adversary B_A for H

INPUT: A hash function H and h' such that $h' \leftarrow \{0,1\}^n$

OUTPUT: x where $h' = H(x)$

$h = P0(h')$

$x = A(H^{P0}, \epsilon, t)(h)$

return x

To find such an x, we require an efficient preimage finding adversary $A(H^{P0}, \epsilon, t)$ where time $t = t'$ + time consumed for calculating $P0$ and t' is the time consumed by adversary B_A. But hash function H is preimage resistant, so existence of such

an adversary B_A is not possible.
$\Rightarrow$ H^{P0} is preimage resistant.

Theorem 3. *If there exists a second preimage finding adversary $A(H^{P0}, \epsilon, t)$ then there also exists a second preimage finding adversary $B_A(H, \epsilon, t')$.*

Proof: Let us consider an efficient adversary $A(H^{P0}, \epsilon, t)$ which when provided with h and x_1, can find x_2 such that

$$h = H^{P0}(x_1) = H^{P0}(x_2).$$

Algorithm 3 Second Preimage adversary B_A for H

INPUT: A hash function H, x_1 and h'
OUTPUT: x_2 where $H(x_1) = H(x_2)$ and $x_1 \neq x_2$

$h = P0(h')$
$x_2 = A(H^{P0}, \epsilon, t)(h, x_1)$
return x_2

To find such an x_2 we require efficient second preimage finding adversary $A(H^{P0}, \epsilon, t)$ where time $t = t'$ + time consumed for calculating $P0$ and t' is the time consumed by adversary B_A. But hash function H is second preimage resistant, so existence of such a B_A is not possible.
$\Rightarrow$ H^{P0} is second preimage resistant.

3.4 HMAC^{P0}-H$^{P0}(K, M)$ is not secure

The attack described in [13] relies on the fact that given large numbers of queries on oracles HMAC-H(K, M) and HMAC-H(K', M), a collision will occur at some instance. After the collision, the same input is forwarded to identical stages in both the cases, so this trend (collisions) will continue. Hence the walks A and B will enter in the same cycle, enabling the attacker to mount various attacks. To prevent these attacks, walks A and B should not enter in the same cycle. Fig. 3.3 illustrates the walk generation using HMAC^{P0}-H$^{P0}(K, M)$ and HMAC^{P0}-H$^{P0}(K', M)$. To avoid the attacker from getting a cycle, occurrence of the same consecutive outputs (either intermediate or final) of HMAC^{P0}-H$^{P0}(K, M)$ and HMAC^{P0}-H$^{P0}(K', M)$ must be prevented. By prepending an extra 0 bit (or byte) in every call to HMAC-H(K, M), the authors of [13] tried to make the internal states different. So that even if the outputs of $H_{K_{out}}$ or $H_{K_{in}}$ collide in HMAC^{P0}-H$^{P0}(K, M)$ and HMAC^{P0}-H$^{P0}(K', M)$, the outputs of next stage will never collide. This is because inputs to the next stage are different for both calls due to the prepended extra 0 (If at some point h_0 and

t_1' collides then 0 is prepended to h_0). So the values at b and c' is not same hence outputs t_1 and h_1' also differ. If we manage to keep the inputs same in both cases then collision will propagate, resulting in cycles. In Fig. 8 we can observe that in this scenario if the collision takes place at any point, then the probability of a collision taking place at the next step is 1.

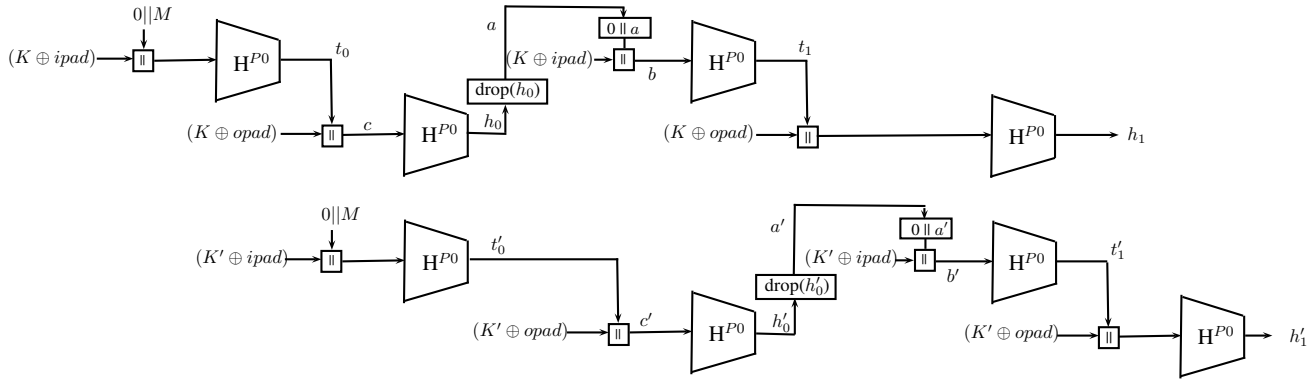

Figure 3.3: Walk Generation using oracle HMAC^{P0}-H$^{P0}(K, M)$ and HMAC^{P0}-H$^{P0}(K', M)$.

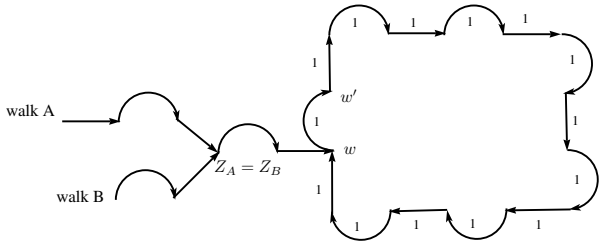

Figure 3.4: Walk A and B entering into same cycle in synchronization i.e. $Z_A = Z_B$. After entering the cycle the probability of obtaining collision at consecutive step is 1 on each edge.

In Fig. 3.3, HMAC^{P0}-H$^{P0}(K, M)$ and HMAC^{P0}-H$^{P0}(K', M)$ are used as oracles, both of which use H^{P0} as underlying hash function. The message M is prepended with 0 in both the cases. The output given by H^{P0} hash is prepended with 0. In the same figure we can observe that due to this step, if a collision occurs at h_0 and t_0' (i.e. $0||h_0 = 0||t_0'$) then it will not propagate to the next step. As all calls to H^{P0} provide 0 prepended to the output of the actual hash function H, due to which at the beginning of next call of HMAC^{P0}-H$^{P0}(K, M)$ or HMAC^{P0}-H$^{P0}(K', M)$, there will be two extra 0 bits (one from the hash function H^{P0} and another one prepended according to the patching scheme $P0$). To tackle this, we have to deploy $drop(x)$ function block which will remove this extra zero. If this extra zero is not removed then the length of b will become 1 bit longer than c' and hence the hash value will differ completely.

Due to the dropping of this extra 0 bit, the value at b and c' becomes the same (i.e. $h_0 = t_0'$). As a result of this, t_1 and h_0' will also collide (i.e. $t_1 = h_0'$). This chain will continue and hence cycles will be obtained by the attacker. So the patching

scheme proposed in [13] is completely broken when a hash function which is collision resistant, preimage resistant and 2nd preimage resistant, but which is not a random oracle.

3.5 HMAC^{P0}-H(K, M) is secure in the Random Oracle Model

The patching scheme proposed in [13] prepends a 0 bit (or byte) and thwarts synchronized computation chain (i.e. cycle). Even if the values collide i.e. $h_0 = t'_1$, since the underlying function H is a random oracle, the probability of having $t_1 = h'_1$ is negligible. The prepended extra 0 increases the length of the message hence altering the output significantly. As a first try, we would like to bring this probability to some measurable bounds.

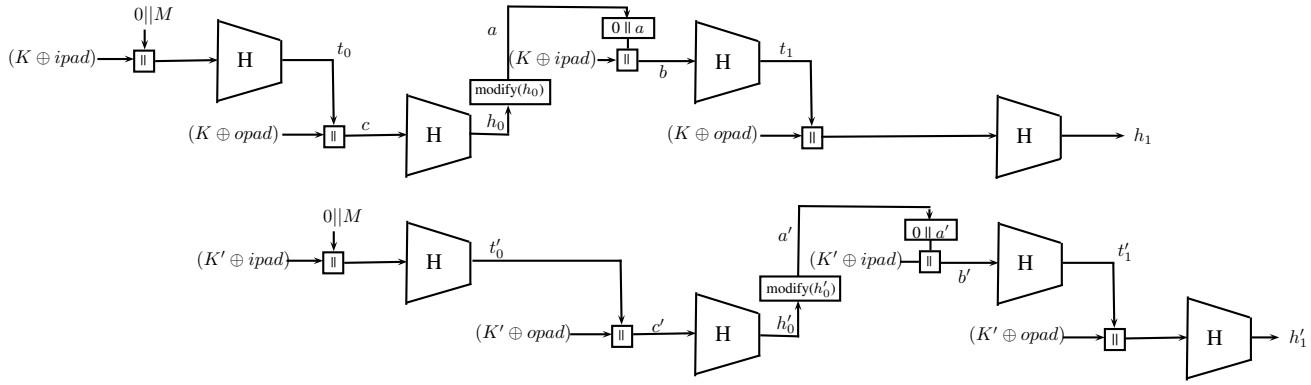

Figure 3.5: Walk Generation using HMAC^{P0} oracles.

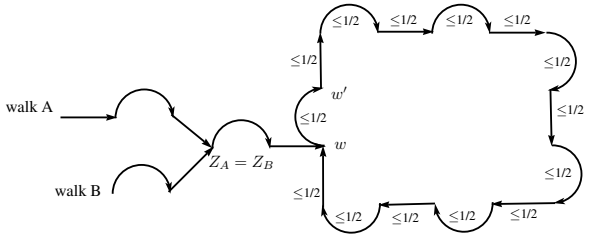

Figure 3.6: Walk A and B entering into same cycle in synchronization i.e. $Z_A = Z_B$. After entering the cycle the probability of obtaining collision at consecutive step is at most 1/2 on each edge.

We propose function $modify(x)$ to do some modifications to x. Suppose that a collision happens at some point in the path i.e. $h_0 = t'_0$, as shown in Fig. 9. After the $modify$ function is applied to h_0, the probability of having a collision in each subsequent step is at most 1/2. In figure 3.6, a cycle is shown with the probability of having collision after each step, i.e., if a collision takes place at w then with probability $\leq 1/2$, a collision will also occur at w'. As at least $2^{n/2}$ elements are needed in a computational chain, the probability of getting a chain will be $\leq 2^{-2^{n/2}}$. This prob-

ability is very low and infeasible in real world scenario i.e. for HMAC-SHA1(K, M) it is $2^{-2^{80}}$. Hence HMAC-H(K, M) is secure when a collision resistant, preimage resistant and second preimage resistant is used which behaves like a random oracle.

3.6 Insecurity of any public and reversible patch

In chapter 3.2 we observed that the attack is possible due to $drop(x)$ function which is the inverse of the patch (Patching scheme prepends 0 to message whereas drop(x) drops the prepended 0). So the attack is only possible when attacker knows the patching scheme and can find its inverse. In this chapter we will demonstrate how HMAC^P-H$^P(K, M)$ scheme is insecure for any patch when patch is public and reversible.

Hash construction H^P

To demonstrate attacks on generic design, hash function H^P is defined such that H^P is based on a good hash function H. Let H^P be a collision, preimage and second preimage resistant function. H^P is a random oracle or not depends on the function P which applies on the output of function H. Let P be a public and reversible function. The hash function H^P is defined as

$$H^P(M) = P(H(M)) = P(h)$$

and construction of H^P is shown in Fig. 3.7

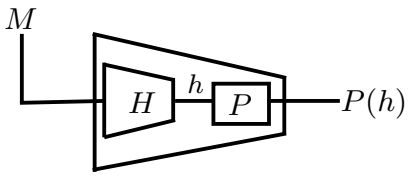

Figure 3.7: The H^P construction.

3.7 Hash H^P is CR, PR, Second PR

As we discussed in chapter 3.2 the security of HMAC-H(K, M) is depicted in terms of unforgeability but depends on the underlying hash function H. We will check the HMAC^{P0}-H(K, M) with the assumption that underlying H is a good hash function.

Theorem 4. *If there exists a collision finding adversary $A(H^P, \epsilon, t)$ then there also exists a collision finding adversary $B_A(H, \epsilon, t')$.*

Proof: Let us consider an efficient adversary $A(H^P, \epsilon, t)$ which can find collisions for H^P with more than negligible probability.

Algorithm 4 Collision finding adversary B_A for H

INPUT: A hash function H

OUTPUT: x_1,x_2 where $H(x_1) = H(x_2)$ but $x_1 \neq x_2$

x_1,$x_2 = A(H^P, \epsilon, t)$

return x_1,x_2

To find a colliding pair for H, we require an efficient collision finding adversary $A(H^P, \epsilon, t)$ where time $t = t'$ + time consumed for calculating P and t' is the time consumed by adversary B_A. But hash function H is collision resistant, so existence of such an adversary B_A is not possible.
$\Rightarrow$ H^P is collision resistant.

Theorem 5. *If there exists a preimage finding adversary $A(H^P, \epsilon, t)$ then there also exists a preimage finding adversary $B_A(H, \epsilon, t')$.*

Proof: Let there be an efficient adversary $A(H^P, \epsilon, t)$ which when provided with h, can find x such that

$$h = H^P(x).$$

Algorithm 5 Preimage finding adversary B_A for H

INPUT: A hash function H and h' such that $h' \leftarrow \{0,1\}^n$

OUTPUT: x where $h' = H(x)$

$h = P(h')$

$x = A(H^P, \epsilon, t)(h)$

return x

To find such an x, we require an efficient preimage finding adversary $A(H^P, \epsilon, t)$ where time $t = t'$ + time consumed for calculating P and t' is the time consumed by adversary B_A. But hash function H is preimage resistant, so existence of such an adversary B_A is not possible.
$\Rightarrow$ H^P is preimage resistant.

Theorem 6. *If there exists a second preimage finding adversary $A(H^P, \epsilon, t)$ then there also exists a second preimage finding adversary $B_A(H, \epsilon, t')$.*

Proof: Let us consider an efficient adversary $A(H^P, \epsilon, t)$ which when provided with

h and x_1, can find x_2 such that

$$h = H^P(x_1) = H^P(x_2).$$

Algorithm 6 Second Preimage adversary B_A for H

INPUT: A hash function H, x_1 and h'
OUTPUT: x_2 where $H(x_1) = H(x_2)$ and $x_1 \neq x_2$

$h = P(h')$
$x_2 = A(H^P, \epsilon, t)(h, x_1)$
return x_2

To find such an x_2 we require efficient second preimage finding adversary $A(H^P, \epsilon, t)$ where time $t = t'$ + time consumed for calculating P and t' is the time consumed by adversary B_A. But hash function H is second preimage resistant, so existence of such a B_A is not possible.
$\Rightarrow$ H^P is second preimage resistant.

3.8 HMAC^P-H$^P(K, M)$ is not secure

To analyse the security of HMAC^P-H$^P(K, M)$, we consider two oracles HMAC^P-H$^P(K, M)$ and HMAC^P-H$^P(K', M)$ depicted in Fig. 3.8. From previous chapters we know that $patch$ part changes the input before feeding it to HMAC-H(K, M) hence thwarting the attack. Even by using custom hash function H^P which outputs $P(h)$ we are not able to get a computational chain. Suppose collision happened at $h_0 = t'_0$, still $t_1 \neq h'_0$. This is due to the fact that b differs from c', because in the pathc P has taken place in the case of oracle HMAC^P-H$^P(K, M)$. Therefore to carry the attack we have to get rid of this extra padding (applied to h_0 in case of HMAC^P-H$^P(K, M)$). As we know P is public and reversible so we can easily construct function P^{-1} which is inverse of P. In Fig. 3.8, the attacker uses P^{-1} to remove extra patching of h_0 such that $h_0 = P(a)$ where $a = P^{-1}(h_0)$. So inputs b and c' become same and hence adversary can get computational chains. Hence HMAC^P-H$^P(K, M)$ construction is not safe if the patch P is public and reversible.

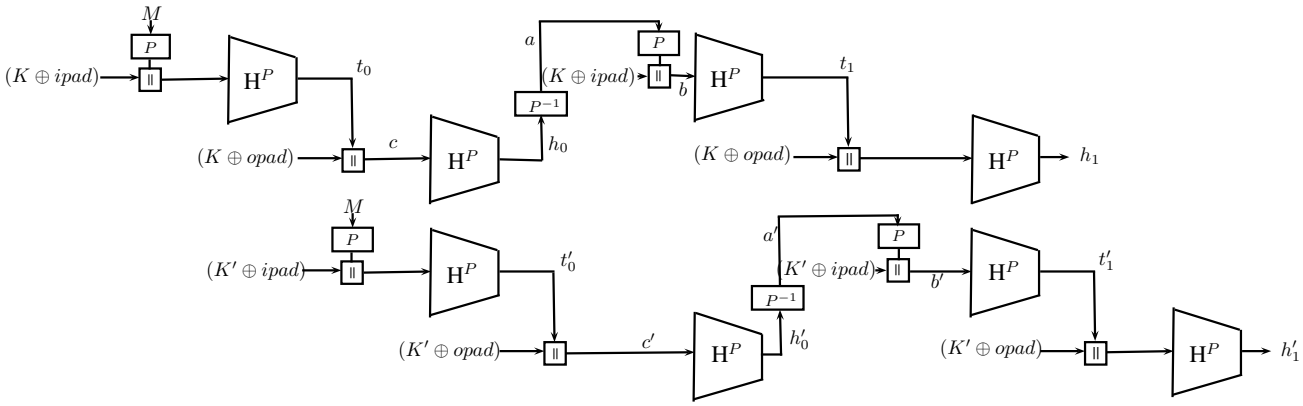

Figure 3.8: Path generation using HMAC^P-H^P oracles.

Chapter 4

Our two new patch proposals

In previous chapters we discussed the patching scheme proposed by [13] and showed that it is not secure against attack described by them. We observed that if one wants to patch the HMAC-H(K, M) scheme then patching scheme should satisfy some minimum conditions. Such a patching scheme should be either secret or one way. We will discuss these and try to give our final proposal.

4.1 Secret Patch

A secret patch is one which is unknown to the attacker, i.e, either the attacker is unaware of the patching scheme or unaware of the patch applied despite knowing the patching scheme. This gives a rough idea that if the patch scheme is kept secret then one can use constant patch such as $patch(M) = M \oplus J$ where J is a constant and is only known to the designer. In this case if the information about J is leaked, the whole scheme will be compromised. So instead of using a secret constant, some other secret should be used which must keep changing with its use, i.e. key K. Further, instead of applying xor to whole message, we can just xor first block of message M of length d.

4.1.1 Secret Patch SP(K, M)

Secret patching scheme SP(K, M) is defined as

$$SP(K, M) = M[1] \oplus K || M[2]M[3] \dots M[s]$$

where message M is divided into s blocks of block length d, say $M[1]M[2]M[2] \dots M[s]$. If $|M| < d$ then $M[1]$ is padded with 0's such that $|M[1]00 \dots 0| = d$, otherwise it is used as it is. K is a secret key of length b-bits. If $b < d$ then K is padded with 0's such

that $|K_2 00\ldots0| = d$, if $b > d$ then $H(K)$ is used as key where H is a hash function with output length d. We will consider K as the padded secret key from now on. In the case of a secret patch we don't bother about the randomness of underlying hash function H, i.e., we need H to be collision resistant, preimage and second preimage resistant but we don't care about its randomness.

Theorem 7. *The HMAC scheme will be secure with respect to related key attacks using cycle detection described in [13] if a secret patch is used. Secret patch refer to patching scheme which is applied to message M before passing it into HMAC and the attacker can't predict $patch$ with more than negligible probability.*

Proof: Secret patch is

$$SP(K, M) = M[1] \oplus K||M[2]M[3]\ldots M[s].$$

HMAC^{SP_K}-H(K, M) is HMAC-H(K, M) which is using secret patch $SP(K, M)$ as the patching scheme, any collision resistant, preimage resistant and second preimage resistant hash function H (not necessarily a random oracle). Here $\overline{K} = K00\ldots$ whereas $|\overline{K}_1| = d$ and M is the message. To analyse the security of HMAC^{SP_K}-H(K, M), in Fig. 4.1, we have path generation by using oracles HMAC^{SP_K}-H(K, M) and HMAC$^{SP_{K'}}$-H(K', M).

As discussed earlier, HMAC^{SP_K}-H(K, M) will behave like a black box. So an attacker can only mount attack between two calls to oracle HMAC^{SP_K}-H(K, M) (or HMAC$^{SP_{K'}}$-H(K', M)). If h_0 and t'_0 collide then for a successful attack b and c' should also collide, so that the collision chain can propagate. In case of HMAC^{SP_K}-H(K, M), h_0 will be applied upon by patch SP(K, M). Therefore, the only way to make b and h_0 same is to apply SP$^{-1}(K, M)$ on h_0 so that when SP(K, M) is applied on it, it remains h_0 i.e. h_0 = SP(K, a) = $K \oplus a[1]||a[2]a[3].....a[s]$.

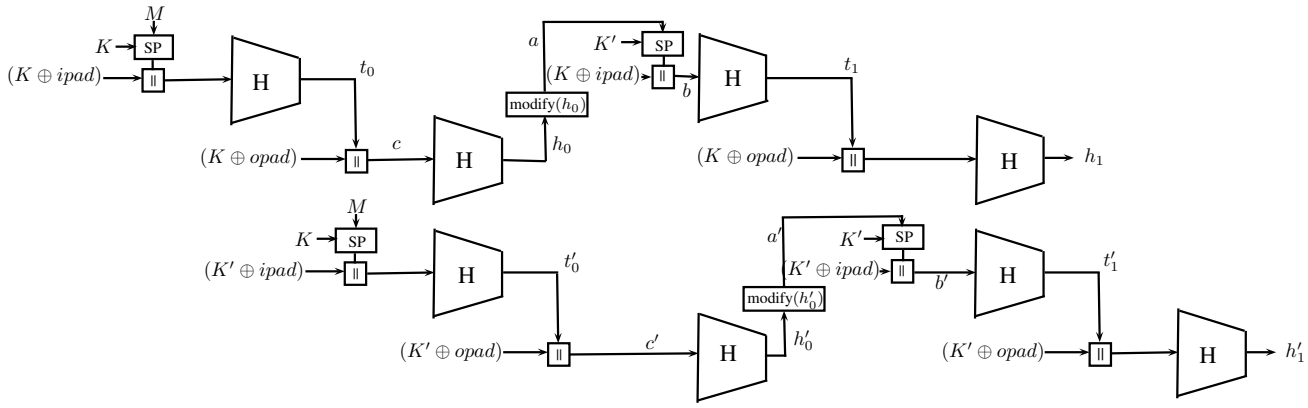

Figure 4.1: Path generation using HMAC^{SP} oracles.

Hence the attacker needs the secret key K to carve such a out of h_0. The attacker attempt to guess the key and guesses K. The probability of guessing the right key

is

$$\text{Prob } [K = h_0 \oplus a] \leq max(2^{-b}, 2^{-d}) \leq \text{Negligible}$$

where the total effort required is $min(2^b, 2^d) + 2^{n/2}$. Note that 2^b (or 2^d) is the effort of getting the key K and $2^{n/2}$ is number of consecutive rounds needed to construct a cycle. As

$$\text{Total Complexity } = min(2^b, 2^d) + 2^{n/2}$$

which is very high, so the probability of getting a synchronized cycle in this case is negligible.

We emphasize that instead of using same key K for secret patch as well as for HMAC if two different keys K_1,K_2 are used i.e. HMAC$^{SP_{K_2}}$-H(K_1, M) and secret patch SP_{K_2}, Then the construction prevents related key attacks due to cycle detection techniques but it allows forgery attack on HMAC(K, M). As shown in Fig. 4.2 if we use two different keys K_1,K_2 when calculating secure tag of message M then tag can be forged by using keys K_1,K_2' on a crafted message M' such that $K_2 \oplus M = K_2' \oplus M'$. When such message, key pair is fed to the construction it will produce the same secure tag h in both the cases.

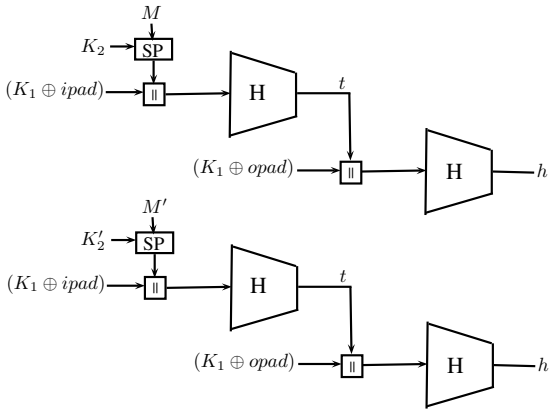

Figure 4.2: Oracles HMAC$^{SP_{K_2}}$-H(K_1, M) and HMAC$^{SP_{K_2'}}$-H(K_1, M')

Therefore, by using this attack, an adversary can forge secure tags. If single key K is used and the attacker tries to forge a secure tag on HMAC^{SP_K}-H(K, M). It is impossible to have two messages M, M' such that $K \oplus M = K \oplus M'$. If the attacker chooses different K for two separate HMAC^{SP_K}-H(K, M) calls then the inner and the outer keys will be different in both the cases. This will prevent forgery attacks on the scheme. So we can not use two different keys for this purpose.

$\Rightarrow$ HMAC scheme is secure against related key attacks described in [13] if the patch is secret.

We propose HMAC^{SP_K}-H(K, M) as secret patch, shown in Fig. 4.3 and defined as

$$\begin{aligned}\text{HMAC}^{SP_K} - H(K,M) &= \text{HMAC}(K, SP(K,M)) \\ &= H_{K_{out}}(H_{K_{in}}(SP(K,M))) \\ &= H_{K_{out}}(H_{K_{in}}(K \oplus M[1]||M[2]M[3]\dots M[s])))\end{aligned}$$

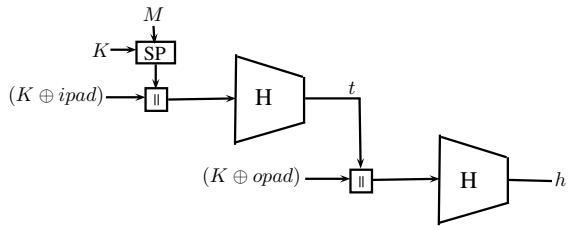

Figure 4.3: The HMAC^{SP_K}-H(K, M) construction

Hence HMAC^{SP_K}-H(K, M) (also denoted as HMAC- H$(pad^{0^*}(K), SP(K, M))$) is secure against cycle detection based related key attacks shown in [13].

4.2 One way Patch Ow(M)

A good one way function is one in which computation in one direction is easy and fast whereas it is very hard (or may be impossible) to go in the other direction. This will be applied to message M before passing it into HMAC-H(K, M) scheme, though it is public but no adversary can efficiently invert its output to obtain the correct input. The one way property of the function is not enough because if the attacker finds collision on this function, he can forge a secure tag. So we have to use a good collision, preimage and 2nd preimage resistant function for the patching scheme.

4.2.1 Collision Resistant One Way Patch CrOw(M)

Collision Resistant One way patching scheme CrOw(M) is defined as

$$CrOw(M) = f'(M[1])||M[2]M[3]\dots M[s]$$

here message M is divided into s blocks of block length d say $M[1]M[2]M[2]\dots M[s]$ and f' is a one way function with output length d. The construction HMAC^{CrOw}-H(K, M) is shown in Figure 4.4

HMAC^{CrOw}-H(K, M) or HMAC-H$(pad^{0^*}(K), CrOw(M))$) is HMAC-H(K, M) construction which is using one way patch $CrOw(K, M)$ as the patching scheme, and any collision resistant, preimage and second preimage resistant hash function H (not necessarily a random oracle) internally. To analyse the security of HMAC^{CrOw}-H(K, M),

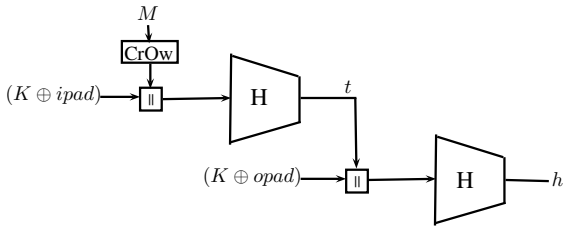

Figure 4.4: The HMAC^{CrOw}-H(K, M) construction

in Fig. 4.5 we show the path generation by using oracles HMAC^{CrOw}-H(K, M) and HMAC^{CrOw}-H(K', M).

As we discussed earlier, HMAC^{CrOw}-H(K, M) will behave like a black box. The attacker will be left with only one choice for mounting the attack, i.e., between two calls to oracle HMAC^{CrOw}-H(K, M) (or HMAC^{CrOw}-H(K', M)). If h_0 and t'_0 collides then for a successful attack b and c' should also collide, so that the collision chain can propagate. In case of HMAC^{CrOw}-H(K, M), the value h_0 will be patched by patch CrOw(M). Therefore, the only way to make b and h_0 same is to apply CrOw$^{-1}(M)$ on h_0 so that when CrOw is applied on it, it remains h_0 i.e. h_0 = CrOw(a) = $f'(a[1])||a[2]a[3].....a[s]$.

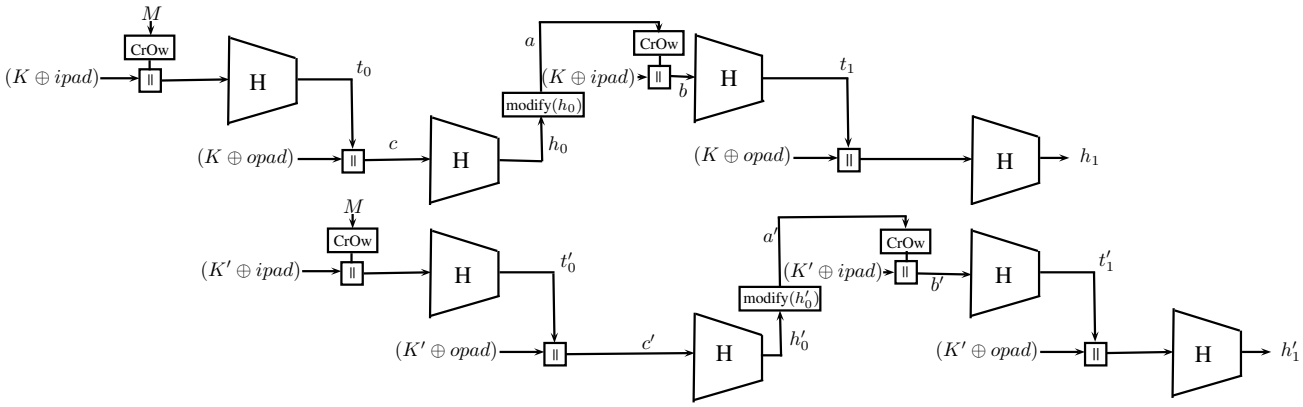

Figure 4.5: Path generation using HMAC^{CrOw} oracles.

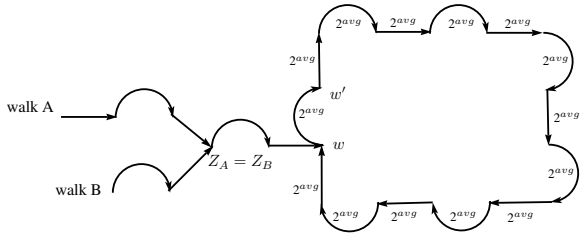

Figure 4.6: Walk A and B entering into same cycle in synchronization i.e. $Z_A = Z_B$. After entering the cycle the probability of obtaining collision at consecutive step is at most 2^{avg} on each edge.

As the patch is a one way function f' hence no attacker can design an inverse

function f'^{-1} for it. To carry out the attack we need to find preimage of the given $f'(M)$ for each step. Let the average complexity to do this be 2^{avg}. Since H is a d bit output hash function and the attacker has to do this for all the steps, a total of $2^{n/2}$ steps are needed to get synchronized cycle as shown in Fig. 4.6. As a result, the total complexity of this attack will be around

$$\text{Total Complexity} \equiv 2^{avg} * 2^{n/2} = 2^{avg+n/2}.$$

Therefore, if a good collision resistant one way function f' is used then the complexity is very high. Hence HMAC^{CrOw}-H(K, M) is very difficult to attack by using cycle detection based attacks discussed in [13].

4.3 Comparison

We have proposed two patches for HMAC. In order to provide our final patch We need to find the best of the two using comparative analysis techniques. Proposed secret patch SP is given by

$$SP(K, M) = M[1] \oplus K||M[2]M[3] \dots M[s].$$

Proposed public one way patch CrOw is

$$CrOw(M) = f'(M[1])||M[2]M[3] \dots M[s].$$

Both of the patches are individually capable of securing the scheme. The security of collision resistant one way patch depends on choice of function $f'(x)$ which can be any good function, we can't provide concrete complexity bounds. Also patch SP(K, M) uses XOR operation on first block of message $M[1]$ and key K whereas the patch CrOw(M) calculates function f' on first block of message $M[1]$, so from efficiency point of view the patch SP(K, M) is better because the XOR operation is lightweight as compared to any good one way function f' where many XOR/other operations may be required to be implemented to achieve randomness and preimage resistance.
$\Rightarrow$ Secret Patch SP(K, M) will be the more efficient one in preventing the attacks described in [13] on HMAC-H(K, M).

Chapter 5

Preventing weak keys based attack with our patches

In [8], the authors described two kinds of weak key pairs which when used with HMAC-H(K, M) can allow an attacker to mount attacks on the scheme. In order to prevent these pairs we propose a new padding for HMAC. This padding will prevent the colliding keys from occurring. The padding is defined below.

Pad$^{10^*}(x)$ padding scheme

or 10^* padding, defined as $pad^{10^*}(x) = x||10^{v-1}$ where $v = d - |x|$.

5.1 Existence of colliding pairs

Table 5.1 shows existence of colliding key pairs in various configurations of HMAC-H(K, M). These pairs exists due to the fact that padding scheme pad^{0^*} pads two keys in such a manner that they become same. But when pad^{10^*} is used, due to the bit 1 in the padding scheme, no two keys of different length can become same (provided that for both $|K| < d$). It is also clear from the table that in case of padding pad^{0^*} colliding pairs exists in every case except when both keys are of length $|K| = d$. However in the case of padding pad^{10^*}, colliding pairs simply don't exist in any case.

5.1.1 Existence of ambiguous pairs

Table 5.1 shows existence of ambiguous key pair in various configurations of HMAC-H(K, M). It is clear from the table that in case of padding pad^{0^*}, key size bound to

avoid ambiguous pairs is $|K| < d - 1$. However in case of padding pad^{10^*} to avoid ambiguous pairs key size should be $|K| < d - 2$.

Lemma 1. *In case of padding pad^{10^*}, there exists no weak key pair only if key of size $|K| < d - 2$ is used.*

Explanation: In standard HMAC-H(K, M) ambiguous pairs only exists when $|K| = d - 1$ or d. Because for keys of size $|K| = d - 1$ and d, due to pad^{0^*} of HMAC-H(K, M) last two bits of the key K will always be 00. Last two bits of constant $ipad$ and $opad$ are 10 and 00 respectively ($ipad = 0x363636 \ldots 36$ having last two bits as 10 and $opad = 0x5C5C5C \ldots 5C$ having last two bits as 00). Hence, the last two bits of inner key $K_{in} = K \oplus ipad$ are 10 and for outer key $K_{out} = K \oplus opad$ are 00. If a key of length $|K| = d - 1$ or d is used, then the last bits of inner and outer keys may come out to be the same. Due to this difference, keys with length $|K| < d - 1$ are safe to use in HMAC-H(K, M).

Similarly last three bits of constant $ipad$ and $opad$ are 110 and 100 respectively. When a key of size $|K| < d - 2$ is used then the last three bits of padded key will be 100. Hence last three bits of inner key $K_{in} = K \oplus ipad$ are 010 and for outer key $K_{out} = K \oplus opad$ are 000. Due to this difference, the problem of ambiguous pairs doesn't arise. On the other hand, if a key of length $|K| \geq d - 2$ is used, then the last bits of inner and outer keys may come out to be the same. That is why bound $|K| < d - 2$ must be followed to avoid ambiguous pairs.

5.2 Security against attacks

Table 5.2 shows the feasibility of attacks on various configurations of HMAC-H(K, M). We have discussed the following two attacks in the table.

Related key attacks based on ambiguous pairs

Related key attacks comprises of distinguishing-R, distinguishing-H, internal state recovery and forgery attacks on HMAC-H(K, M) scheme. We have already discussed these in Sec. 2.2.

5.2.1 Indifferentiability attacks based on colliding pairs

As described in [8], indifferentiability attack on HMAC-H(K, M) is said to be performed successfully if the attacker can distinguish between pair of oracles consisting of HMAC-H(K, M) with underlying hash H and a random oracle with a simulator

Construction	Key Size $\|K\|$	Colliding key pairs	Ambiguous key pairs	Construction	Key Size $\|K\|$	Colliding key pairs	Ambiguous key pairs
HMAC-H$(pad^{0^*}(K), M)$ [2]	$0 < \|K\| < d-1$	O	X	HMAC-H$(pad^{10^*}(K), M)$ [our contribution]	$0 < \|K\| < d-1$	X	O
	$d-1 \leq \|K\| < d$	O	O		$d-1 \leq \|K\| < d$	X	O
	$0 < \|K\| < d-2$	O	X		$0 < \|K\| < d-2$	X	X
	$d-2 \leq \|K\| < d$	O	O		$d-2 \leq \|K\| < d$	X	O
	$\|K\| = d$	X	O		$\|K\| = d$	X	O
HMAC-H$(pad^{0^*}(K), P0(M))$ [13]	$0 < \|K\| < d-1$	O	X	HMAC-H$(pad^{10^*}(K), P0(M))$ [our contribution]	$0 < \|K\| < d-1$	X	O
	$d-1 \leq \|K\| < d$	O	O		$d-1 \leq \|K\| < d$	X	O
	$0 < \|K\| < d-2$	O	X		$0 < \|K\| < d-2$	X	X
	$d-2 \leq \|K\| < d$	O	O		$d-2 \leq \|K\| < d$	X	O
	$\|K\| = d$	X	O		$\|K\| = d$	X	O
HMAC- H$(pad^{0^*}(K), SP(K, M))$ [our contribution]	$0 < \|K\| < d-1$	O	X	HMAC- H$(pad^{10^*}(K), SP(K, M))$ [our contribution]	$0 < \|K\| < d-1$	X	O
	$d-1 \leq \|K\| < d$	O	O		$d-1 \leq \|K\| < d$	X	O
	$0 < \|K\| < d-2$	O	X		$0 < \|K\| < d-2$	X	X
	$d-2 \leq \|K\| < d$	O	O		$d-2 \leq \|K\| < d$	X	O
	$\|K\| = d$	X	O		$\|K\| = d$	X	O
HMAC- H$(pad^{0^*}(K), CrOw(M))$ [our contribution]	$0 < \|K\| < d-1$	O	X	HMAC- H$(pad^{10^*}(K), CrOw(M))$ [our contribution]	$0 < \|K\| < d-1$	X	O
	$d-1 \leq \|K\| < d$	O	O		$d-1 \leq \|K\| < d$	X	O
	$0 < \|K\| < d-2$	O	X		$0 < \|K\| < d-2$	X	X
	$d-2 \leq \|K\| < d$	O	O		$d-2 \leq \|K\| < d$	X	O
	$\|K\| = d$	X	O		$\|K\| = d$	X	O

Table 5.1: Comparative summary of existence or non-existence of colliding keys and ambiguous keys when HMAC is used with our patches i.e. $SP(K, M)$ and $CrOw(M)$, and padding schemes i.e. pad^{0^*} and pad^{10^*}. $|K|$ is length of the key K, d is the block size of H. HMAC-H$(pad^x(K), M)$ is HMAC-H(K, M) with pad^x (either pad^{0^*} or pad^{10^*}). Similarly HMAC-H$(pad^x(K), P0(M))$ is for HMAC-H(K, M) patched with patch $P0$ and padding scheme pad^x, HMAC- H$(pad^x(K), SP(K, M))$ is for HMAC-H(K, M) patched with patch $SP(K, M)$ and padding scheme pad^x and HMAC-H$(pad^x(K), CrOw(M))$ is for HMAC-H(K, M) patched with patch $CrOw(M)$ and padding scheme pad^x. Each O entry represent the existence of respective weak key whereas X entry represent the non-existence of respective related keys for given range of key length $|K|$.

Construction	Key Size $\|K\|$	Cycle Detection attack based on ambiguous pairs	Construction	Key Size $\|K\|$	Cycle Detection attack based on ambiguous pairs
HMAC-H$(pad^{0^*}(K), M)$ [2]	$0 < \|K\| < d-1$	X	HMAC-H$(pad^{10^*}(K), M)$ [our contribution]	$0 < \|K\| < d-1$	O
	$d-1 \leq \|K\| < d$	O		$d-1 \leq \|K\| < d$	O
	$0 < \|K\| < d-2$	X		$0 < \|K\| < d-2$	X
	$d-2 \leq \|K\| < d$	O		$d-2 \leq \|K\| < d$	O
	$\|K\| = d$	O		$\|K\| = d$	O
HMAC-H$(pad^{0^*}(K), P0(M))$ [13]	$0 < \|K\| < d-1$	X	HMAC-H$(pad^{10^*}(K), P0(M))$ [our contribution]	$0 < \|K\| < d-1$	O
	$d-1 \leq \|K\| < d$	O		$d-1 \leq \|K\| < d$	O
	$0 < \|K\| < d-2$	X		$0 < \|K\| < d-2$	X
	$d-2 \leq \|K\| < d$	O		$d-2 \leq \|K\| < d$	O
	$\|K\| = d$	O		$\|K\| = d$	O
HMAC- H$(pad^{0^*}(K), SP(K, M))$ [our contribution]	$0 < \|K\| < d-1$	X	HMAC- H$(pad^{10^*}(K), SP(K, M))$ [our contribution]	$0 < \|K\| < d-1$	X
	$d-1 \leq \|K\| < d$	X		$d-1 \leq \|K\| < d$	X
	$0 < \|K\| < d-2$	X		$0 < \|K\| < d-2$	X
	$d-2 \leq \|K\| < d$	X		$d-2 \leq \|K\| < d$	X
	$\|K\| = d$	X		$\|K\| = d$	X
HMAC- H$(pad^{0^*}(K), CrOw(M))$ [our contribution]	$0 < \|K\| < d-1$	X	HMAC- H$(pad^{10^*}(K), CrOw(M))$ [our contribution]	$0 < \|K\| < d-1$	X
	$d-1 \leq \|K\| < d$	X		$d-1 \leq \|K\| < d$	X
	$0 < \|K\| < d-2$	X		$0 < \|K\| < d-2$	X
	$d-2 \leq \|K\| < d$	X		$d-2 \leq \|K\| < d$	X
	$\|K\| = d$	X		$\|K\| = d$	X

Table 5.2: Summary of feasibility of cycle detection based attacks based on ambiguous keys when HMAC is used with our patches i.e. $SP(K, M)$ and $CrOw(M)$, and padding schemes i.e. pad^{0^*} and pad^{10^*}. $|K|$ is length of the key K, d is the block size of H. HMAC-H$(pad^x(K), M)$ is HMAC-H(K, M) with pad^x (either pad^{0^*} or pad^{10^*}). Similarly HMAC-H$(pad^x(K), P0(M))$ is for HMAC-H(K, M) patched with patch $P0$ and padding scheme pad^x, HMAC- H$(pad^x(K), SP(K, M))$ is for HMAC-H(K, M) patched with patch $SP(K, M)$ and padding scheme pad^x and HMAC-H$(pad^x(K), CrOw(M))$ is for HMAC-H(K, M) patched with patch $CrOw(M)$ and padding scheme pad^x. Each O entry represent that respective construction is not secure against that attack whereas X entry represent that respective scheme is secure against that attack.

based on the random oracle. In [11], the authors show that if a component S is indifferentiable from T, then the security of any cryptosystem $C(T)$ based on T is not affected when T is replaced by S.

For HMAC-H$(pad^{0*}(K), M)$ collision pair based attacks is not possible for $|K| = d$ whereas for all other possible keys it is feasible. Further, ambiguous pair based attacks will only be prevented when $|K| < d - 1$. On the other hand, when used with $pad^{10^*}(K)$, collision pair based attacks are not possible for any case and ambiguous pair based attacks will work for all keys of size except $|K| < d - 2$. Results are similar for HMAC-H$(pad^x(K), P0(M))$. For HMAC- H$(pad^x(K), SP(K, M))$ and HMAC-H$(pad^x(K), CrOw(M))$ when $pad^{10^*}(K)$ is used, colliding pair based attack is feasible for all keys of size except $|K| = d$. On the other hand, when $pad^{10^*}(K)$ is used, colliding pair based attacks are not feasible for any key size. But both constructions are safe from ambiguous key pair based attacks. Therefore to conclude, when the 0 padding scheme, i.e. pad^{0^*} is used with constructions then indifferentiability attacks always exist. But when 10 padding scheme, i.e. pad^{10^*} is used with constructions then schemes HMAC-H$(pad^{0^*}(K), SP(K, M)$ and HMAC-H$(pad^{0^*}(K), CrOw(M))$ become secure against both the attacks, i.e. indifferentiability attacks using colliding pairs and cycle detection attacks based on related keys.

Chapter 6

Conclusions and Future work

In this work, we have shown that HMAC-H(K, M) patched with the patching scheme proposed by Peyrin *et al.* in [13] fails when a collision, preimage and 2nd preimage resistant but not a random oracle function is used as the underlying hash function. We provided the explanation of failure for their patching scheme and showed that the use of secret or collision resistant one way patching scheme (i.e. SP(K, M) and CrOw(M)) can secure HMAC-H(K, M). We proposed that HMAC patched with any one of our two patches with pad^{10} or $10 \ldots 0$ padding, is resistant to cycle detection based generic related key attacks due to ambiguous keys discussed by Peyrin *et al.* [13] and indifferentiability attacks based on colliding keys discussed by Dodis *et al.* [8].

The main advantages of our patches are:

- These are very efficient in comparison to increasing the internal state of hash function used in HMAC.
- These can be easily applied as a wrapper on present implementations.

The attacks are very specific, hence we won't be able to provide security proofs for our patches. But, we have provided explanations for the same. We will try to improve our patches and also explore new designs in future.

Bibliography

[1] Request For Comments: 3174, US Secure Hash Algorithm 1 (SHA1), 2001. IETF Working group.

[2] Bellare, M. New Proofs for NMAC and HMAC: Security without Collision-Resistance. In *[15]* (2006), C. Dwork, Ed., vol. 4117 of *Lecture Notes in Computer Science*, Springer, pp. 602--619.

[3] Bellare, M., Canetti, R., and Krawczyk, H. Keying Hash Functions for Message Authentication. In *CRYPTO* (1996), N. Koblitz, Ed., vol. 1109 of *Lecture Notes in Computer Science*, Springer, pp. 1--15.

[4] Bellare, M., and Kohno, T. Hash function balance and its impact on birthday attacks. In *EUROCRYPT* (2004), pp. 401--418.

[5] Brassard, G., Ed. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings* (1990), vol. 435 of *Lecture Notes in Computer Science*, Springer.

[6] Coron, J.-S., Dodis, Y., Malinaud, C., and Puniya, P. Merkle-damgård revisited: How to construct a hash function. In *CRYPTO* (2005), V. Shoup, Ed., vol. 3621 of *Lecture Notes in Computer Science*, Springer, pp. 430--448.

[7] Damgård, I. B. A design principle for hash functions. In *Proceedings on Advances in Cryptology* (New York, NY, USA, 1989), CRYPTO '89, Springer-Verlag New York, Inc., pp. 416--427.

[8] Dodis, Y., Ristenpart, T., Steinberger, J. P., and Tessaro, S. To Hash or Not to Hash Again? (In)Differentiability Results for H^2 and HMAC. In *CRYPTO* (2012), pp. 348--366.

[9] Leurent, G., Peyrin, T., and Wang, L. New Generic Attacks against Hash-Based MACs. In *ASIACRYPT (2)* (2013), K. Sako and P. Sarkar, Eds., vol. 8270 of *Lecture Notes in Computer Science*, Springer, pp. 1--20.

[10] M. Bellare, R. C., and Krawczyk, H. Request For Comments: 2104, HMAC: Keyed-Hashing for Message Authentication, 1997. IETF Working group.

[11] Maurer, U. M., Renner, R., and Holenstein, C. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *TCC* (2004), M. Naor, Ed., vol. 2951 of *Lecture Notes in Computer Science*, Springer, pp. 21--39.

[12] Merkle, R. C. One Way Hash Functions and DES. In Brassard [5], pp. 428--446.

[13] Peyrin, T., Sasaki, Y., and Wang, L. Generic Related-Key Attacks for HMAC. In Sako and Wang [15], pp. 580--597.

[14] Rivest, R. L. Request For Comments: 1320, The MD5 message-digest algorithm, 1992. IETF Working group.

[15] Sako, K., and Wang, X., Eds. *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings* (2012), vol. 7658 of *Lecture Notes in Computer Science*, Springer.