Forging Attacks on two Authenticated Encryption Schemes COBRA and POET

Mridul Nandi

Indian Statistical Institute, Kolkata, India
mridul.nandi@gmail.com

Abstract. In FSE 2014, an authenticated encryption mode COBRA [4], based on pseudorandom permutation (PRP) blockcipher, and POET [3], based on Almost XOR-Universal (AXU) hash and strong pseudorandom permutation (SPRP), were proposed. Few weeks later, COBRA mode and a simple variant of the original proposal of POET (due to a forging attack [13] on the original proposal) with AES as an underlying blockcipher, were submitted to CAESAR, a competition [1] of authenticated encryption (AE). In this paper, we show a forging attack on the mode COBRA based on any n-bit blockcipher. Our attack on COBRA requires about $O(n)$ queries with success probability of about $1/2$. This disproves the claim proved in the FSE 2014 paper. We also show both privacy and forging attack on the parallel version of POET, denoted POET-m. In case of the modes POET and POE (the underlying modes for encryption), we demonstrate a distinguishing attack making only one encryption query when we instantiate the underlying AXU hash function with some other AXU hash function, namely a uniform random involution. Thus, our result violates the designer's main claim (Theorem 8.1 in [1]). However, the attacks can not be extended to the specifications of POET submitted to the CAESAR competition.

Keywords: Authenticated Encryption, COBRA, POET, Distinguishing Attack and Forging Attack.

1 Introduction

The common application of cryptography is to implement a secure channel between two or more users and then to exchange information over that channel. These users can initially set up their one-time shared key. Otherwise, a typical implementation first calls a key-exchange protocol for establishing a shared key or a session key (used only for the current session). Once the users have a shared key, either through the initial key set-up or key-exchange, they use this key to authenticate and encrypt the transmitted information using efficient symmetric-key algorithms such as a *message authentication code* $\mathsf{Mac}(\cdot)$, *pseudorandom function* $\mathsf{Prf}(\cdot)$ and (possibly tweakable symmetric-key) *encryption* $\mathsf{Enc}(\cdot)$ respectively. The encryption Enc provides privacy or confidentiality of the *plaintext* M. The message authentication code Mac and pseudorandom function

Prf provide data-integrity authenticating the transmitted message $(M,\ A)$, a pair of plaintext M and an *associated data* A. Mac also provides user-authenticity (protecting from impersonation). An **Authenticated Encryption** scheme (or simply **AE**) serves both of the purposes in an integrated manner. An authenticated encryption scheme AE has two functionalities one of which, called tagged encryption, essentially combines message authentication code and encryption, and the other combines verification and decryption algorithms.

1. **Tagged encryption** $\mathsf{AE}.\mathrm{enc}_k$: On an input message M from a message space $\mathcal{M} \subseteq \{0,1\}^*$ and an associated data A from an associated data space $\mathcal{D} \subseteq \{0,1\}^*$, it returns a **tagged ciphertext**[1] $Z \in \{0,1\}^*$.
2. **Verified decryption** $\mathsf{AE}.\mathrm{dec}_k$: On an input tagged ciphertext Z and an associated data A, it returns a plaintext M when $Z = \mathsf{AE}.\mathrm{enc}_k(M, A)$, called valid tagged ciphertext. Otherwise, for all invalid tagged ciphertext Z, it returns a special symbol $\perp$.

Note that both algorithms take the shared key k from a keys-space $\mathcal{K} = \{0,1\}^{L_{\mathrm{key}}}$ where L_{key} denotes the key-size. The key includes keys for an underlying blockcipher, masking keys etc. Some constructions derive more keys by invoking the blockcipher with different constant inputs.

Privacy and Authenticity Advantage. Informally speaking, an AE scheme is said to have *privacy* if the tagged ciphertext behave like a uniform random string for any adaptively chosen plaintext. More formally, let A be an oracle adversary which can make queries to AE.enc adaptively. Let \$ be a random oracle which returns a uniform random string for every new query. We define the *privacy advantage* of A against AE to be

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{AE}}(A) := \left|\mathsf{Pr}[A^{\mathsf{AE}.\mathrm{enc}_K} = 1] - \mathsf{Pr}[A^{\$} = 1]\right|$$

where the two probabilities are taken under \$, K (usually chosen uniformly from the key-space $\mathcal{K}$) and the random coins of A. Similarly, we define the *authenticity advantage* of A as

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathsf{AE}}(A) := \mathsf{Pr}[A^{\mathsf{AE}.\mathrm{enc}_K} = Z,\ \mathsf{AE}.\mathrm{dec}_K(Z) \neq \perp \text{ and } Z \text{ is fresh}]$$

where the probability is taken under K and the random coins of A. By fresh, we mean that Z is not a response of an encryption query of A.

1.1 Two AE Schemes COBRA and POET submitted to CAESAR

CAESAR [1] is an ongoing competition for authenticated encryption schemes. The final goal of the competition is to identify a portfolio of different authenticated encryption schemes depending on different applications and environments. Fifty seven schemes have been submitted. AES-COBRA and POET are two such

[1] A tagged ciphertext usually consists of a ciphertext and tag. However, there may not exist a clear separation between ciphertext and tag.

submissions. Variants of these two schemes have been published before in FSE 2014. In [13] Guo et al. demonstrated a forging attack against POET making only one encryption query. So the designers of POET modified it accordingly to resist this forging attack and submitted the revised version to CAESAR.

1.2 Our Contribution

In this paper, we investigate the resistance of the two authenticated encryption schemes COBRA and POET against forging and privacy attacks. The paper is essentially divided in two sections: Section 4 describes the forging attack for COBRA and Section 5 describes forging and privacy analysis on the POET-mode and its parallel variant, called POET-m.

1. **Attack on COBRA**. In this paper, we show a forging attack on the submitted version of AES-COBRA. In fact, the attack works for the mode COBRA based on any blockcipher. Thus it disproves the claim stated in [4]. The authenticity advantage of our proposed algorithm is about $1/2$ and it makes about $2n$ encryption queries where n is the plaintext size of the underlying blockcipher.
2. **Analysis of POET and POET-m**. The designers of POET have recommended a parallel version, denoted POET-m. We provide distinguishing and forging attacks on it. Moreover, the designers claimed security of POET for an arbitrary AXU or almost XOR universal hash function (the formal definition of AXU hash function is given in Section 2). Here we disprove their claim by showing a distinguishing attack on a special choice of AXU, namely a uniform random involution. Thus, the security proof of the claims have flaws. We also extend this to a forging attack. All these attack algorithms make very few encryption queries and succeed with probability close to one.

We would like to note that while the COBRA is affected by our attack, the instantiation of the POET candidate which uses specific AXU hash functions is not affected.

2 Basics of Almost XOR Universal (AXU) Hash

2.1 Notation and Basics

In this paper, we fix a positive integer n which denotes the block size of the underlying blockcipher. We mostly use AES (advanced encryption standard) [11] with 128 bit key size as the underlying blockcipher and in this case $n = 128$.

Binary Field. We identify the set $\{0,1\}^n$ as the binary field of size 2^n. An n bit string $\alpha = \alpha_0\alpha_1...\alpha_{n-1}$, $\alpha_i \in \{0,1\}$ can be equivalently viewed as a polynomial $\alpha(x) = \alpha_0 + \alpha_1 x + \cdots + \alpha_{n-1}x^{n-1}$. For notational simplicity, we write the concatenation of two binary strings α and β as $\alpha\beta$. The field addition between two n bit strings is bit-wise addition $\oplus$ (we also use "+"). Let us fix a primitive

polynomial $p(x)$ of degree n. Field multiplication between two n-bit strings α and β can be defined as the binary string corresponding to the polynomial $\alpha(x)\beta(x)$ mod $p(x)$. We denote the multiplication of α and β as $\alpha \cdot \beta$. Thus, the zero polynomial $\mathbf{0}$ and the constant $\mathbf{1}$ polynomial are the additive and multiplicative identity respectively. Moreover, x is a primitive element since the polynomial $p(x)$ is primitive.

2.2 Almost XOR Universal (AXU) Hash

Universal hash functions and its close variants *strongly universal*, *AXU-hash* [9, 12, 20, 22, 23, 21, 18] are information theoretic notions which are used as building blocks of several cryptographic constructions, e.g., *message authentication code* [9, 24], domain extension of pseudorandom function [5, 8], extractor [17], quasi-randomness and other combinatorial objects [12, 21].

Definition 1 (AXU Hash Function). *A function family $F_L : \mathcal{M} \to \{0,1\}^n$ indexed by $L \in \mathcal{L}$ is called ϵ-AXU [18] if for all $x \neq x' \in \mathcal{M}$ and $\delta \in \{0,1\}^n$, $\mathsf{Pr}_L[F_L(x) \oplus F_L(x') = \delta] \leq \epsilon$ where L is chosen uniformly from $\mathcal{L}$.*

Examples Field Multiplier. Let $L \in \{0,1\}^n$ be chosen uniformly then $F_L(x) = L \cdot x$ (field multiplication on $\{0,1\}^n$) is 2^{-n}-AXU.

Polynomial Hash. Polynomial hash [20] is one of the popular universal hash which can be computed efficiently by Horner's rule [14] (same as computation of CBC message authenticated code [2, 6]).

Definition 2. *[20] We define the polynomial-hash indexed by $L \in \{0,1\}^n$ over the domain $(\{0,1\}^n)^+ := \cup_{i=1}^{\infty}\{0,1\}^{ni}$ as*

$$\mathsf{poly}_L(a_d, a_{d-1}, \ldots, a_0) = a_0 + a_1 \cdot L \cdots + a_{d-1} \cdot L^{d-1} + a_d \cdot L^d$$

where $a_0, a_1, \ldots, a_d \in \{0,1\}^n$ and L^i denotes $L \cdot L \cdots L$ (i times).

It is easy to see that the function mapping $(a_1, \ldots, a_d)$ to $a_1 \cdot L + \cdots + a_d \cdot L^d$ is $\frac{d}{2^n}$-AXU hash function over the domain $(\{0,1\}^n)^d$. To see this, let $(a_1, \ldots, a_d) \neq (b_1, \ldots, b_d)$ and $c \in \{0,1\}^n$. So,

$$c + (a_1 - b_1) \cdot L + \cdots + (a_d - b_d) \cdot L^d$$

is a non-zero polynomial and hence it has at most d distinct roots of L.

Four-Round AES. The AES (for 128 bit keys) has ten rounds. However, it has been shown that four-round AES has good differential properties. More formally, Daemen et al. in [10] showed that four-round AES is a family of 2^{-113}-AXU under the simplified assumption that all four round keys are uniform and independent.

Uniform Random Involution. The uniform random function from $\{0,1\}^n$ to itself is an 2^{-n}-AXU hash function. A function $f : \{0,1\}^n \to \{0,1\}^n$ is

called an **involution** if f is inverse of itself (so it must be permutation). Let I_n denote a random involution whose responses are defined according to the following procedure: After responding to every query, it updates two sets: the set of all queries D and the set of all responses R. On a query $x \notin D \cup R$, it returns an element chosen uniformly from the set $\{0,1\}^n \setminus (D \cup R)$. If $x \in D$ then it returns the previous response corresponding to x. Similarly, if $x \in R$ then it returns the previous query $y \in D$ for which the response was x.

Lemma 1. *The uniform random involution I_n (as defined above) is an $\frac{2}{2^n-2}$-AXU hash function.*

Proof. Let $x \neq x' \in \{0,1\}^n$ and $\delta \in \{0,1\}^n$. Let us assume that $x \oplus x' \neq \delta$. By conditioning $I_n(x) = y$, we must have $I_n(x') = y \oplus \delta$ which happens with probability at most $1/(2^n - 2)$. Note that if $y = x'$ or $y = x \oplus \delta$ then the probability is zero. So $\mathsf{Pr}[I_n(x) \oplus I_n(x') = \delta] \leq \frac{1}{2^n-2}$. Now assume $x \oplus x' = \delta$. So $\mathsf{Pr}[I_n(x) = x'] \leq \frac{1}{2^n-2}$. When $I_n(x) \neq x'$, by similar argument as before we also have differential probability bounded above by $\frac{1}{2^n-2}$. This proves the Lemma. □

2.3 Combination of AXU hash functions

Compositions of AXU hash functions Now we show that property of being an AXU-hash function does not preserve under composition with same key. In other words, there exists an ϵ-AXU F_L for a "small" ϵ such that $F_L \circ F_L$ is not even δ-AXU for any $\delta < 1$. In particular, if we choose F_L to be a uniform random involution then F_L is $\frac{1}{2^n-2}$-AXU whereas the composition $F_L \circ F_L$ is the identity function. Trivially, a similar result holds if we apply the CBC mode for a uniform random involution I_n. The CBC mode applied to a function f is defined as follows:

$$CBC^f(x_1, \ldots, x_d) = y_d, \text{ where } y_i = f(y_{i-1} \oplus x_i), 1 \leq i \leq d$$

and $y_0 = 0^n$. So when $d = 2$ and $x_2 = 0$, $CBC^{I_n}(x_1, 0) = I_n(I_n(x_1)) = x_1$ and so CBC^{I_n} is not δ-AXU for any $\delta < 1$. However, it is true for some specific choices of F_L, e.g. when F_L is field multiplier. In this case, CBC^{F_L} is nothing but the poly-hash which has been shown to be $d/2^n$-AXU (see the paragraph immediately after Definition 2).

Sum of AXU hash functions Now we consider another method of domain extension of AXU hash function. Given an ϵ-AXU F_L, we define the sum hash

$$F_L^{\text{sum}}(x_1, \ldots, x_\ell) = F_L(x_1) \oplus \cdots \oplus F_L(x_\ell).$$

Note that if F_L is linear (which is true for the field multiplier) then the sum hash can be simplified as $F_L^{\text{sum}}(x_1, \ldots, x_\ell) = F_L(x_1 \oplus \ldots \oplus x_\ell)$ for which a collision can be found easily. So it can not be δ-AXU for any $\delta < 1$. However, this does not work when we consider a uniform random function or involution and we concatenate a counter to message blocks.

3 Description of COBRA

COBRA is an authenticated encryption mode based on blockcipher. It was originally published in FSE 2014 [4]. Later the same mode with AES as the underlying blockcipher, called AES-COBRA, was submitted to CAESAR [1]. The mode can be viewed as hash then ECB (or electronic code-book) type encryption where hash function is poly-hash and ECB is applied on a double block, i.e., $2n$ bit plaintext. The double block encryption is defined by two-round Feistel structure [15].[2] As it uses Feistel structure, it is inverse-free. In other words, even though it is based on AES blockcipher, the decryption of COBRA does not require AES decryption.

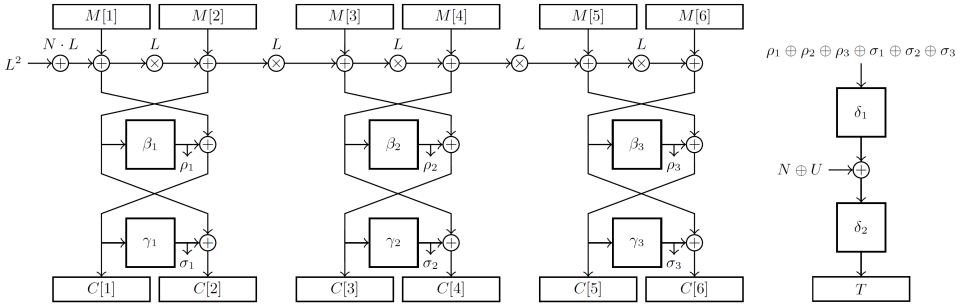

Fig. 3.1. COBRA Modes for ciphertext and tag generation for three double blocks message. U is obtained from associated data, N is nonce and L is the hash key.

3.1 Encryption Mode for COBRA

COBRA is defined for any messages of size at least n bits. Now we briefly describe how the encryption algorithm of COBRA works for all inputs $M \in (\{0,1\}^{2n})^+$. In addition to a message M, it also takes a nonce $N \in \{0,1\}^n$ and an associated data A, and outputs a tagged ciphertext (C, T) where $|C| = |M|$ and $T \in \{0,1\}^n$. Readers are referred to [4, 1] for complete description of the algorithm (i.e., how it behaves for other input sizes). We write $M = M_1 \| \cdots \| M_d$ for some positive integer d where $M_1, \ldots, M_d \in \{0,1\}^{2n}$. We also write $M_i = (M_i[1], M_i[2])$ where $M_i[1], M_i[2] \in \{0,1\}^n$ are also called blocks and M_i's are called **double blocks**. Let β_i' and γ_i's be independent uniform random (or pseudorandom) permutations over $\{0,1\}^n$ for all $i \geq 1$. We describe the COBRA-mode based on

[2] The 3 and 4 rounds security analysis is given in [15] (see [16] for characterization of Luby-Rackoff constructions).

these permutations.[3] It uses the two-round Feistel structure which is defined as follows:

$$\mathsf{LR}_i(X[1], X[2]) = (Y[1], Y[2]), \quad X[1], X[2] \in \{0,1\}^n$$

where

1. $Y[1] = X[1] \oplus \beta_i(X[2])$ and
2. $Y[2] = X[2] \oplus \gamma_i(X[1])$.

It is easy to see that it is invertible and the inverse function $2\mathsf{LR}_i^{-1}(Y[1], Y[2]) = (X[1], X[2])$ where $X[2] = \gamma_i(Y[1]) \oplus Y[2]$ and $X[1] = \beta_i(X[2]) \oplus Y[1]$.

Algorithm: COBRA Encryption
Input: $(M_1[1], M_1[2], \ldots, M_d[1], M_d[2]) \in (\{0,1\}^n)^{2d}$, $N \in \{0,1\}^n$
Output: $(C_1, C_2, ..., C_d) \in (\{0,1\}^{2n})^d$

---

for $i = 1$ to d
  $P_i[1] = \mathsf{poly}_L(1, N, M_1[1], M_1[2], \ldots, M_i[1])$;
  $P_i[2] = \mathsf{poly}_L(1, N, M_1[1], M_1[2], \ldots, M_i[1], M_i[2])$;
  $C_i = \mathsf{LR}_i(P_i[1], P_i[2])$;
end for loop
Return $(C_1, C_2, ..., C_d)$

Algorithm 1: COBRA encryption algorithm for a nonce $N \in \{0,1\}^n$, and a messages M of sizes multiple of $2n$. Note that the associated data has no influence on the ciphertext. It is used for computing the tag.

3.2 Tag Generation and Verified Decryption Algorithm

The final tag T is computed from nonce N and U (depends only on the associated data A) and

$$S := \bigoplus_{i=1}^{d} (P_i[1] \oplus P_i[2] \oplus C_i[1] \oplus C_i[2]).$$

We simply denote the tag by $T(N, U, S)$. One can find the details of the construction of T in [4, 1]. The verified decryption algorithm takes a tagged ciphertext $(C_1, \ldots, C_d, T)$ where C_i's are double blocks and $T \in \{0,1\}^n$. It works as follows:

1. It first computes $P_i = \mathsf{LR}_i^{-1}(C_i)$, $1 \le i \le d$.
2. It returns $\perp$ if $T \neq T(N, U, S)$.
3. Else it returns $(M_1, \ldots, M_d)$ where $M_i[2] = L \cdot P_i[1] \oplus P_i[2]$ and $M_i[1] = L \cdot P_{i-1}[2] \oplus P_i[1]$, $1 \le i \le d$.

[3] These are actually derived from a single blockcipher using the standard masking algorithm (i.e., XEX construction [19]).

4 Forging Attack on COBRA

We first state the following fact which plays key role in our forging attack.

Fact 1.[7] *Let $h \in \{0,1\}^n$ be a fixed element and $h_0^1, h_1^1, \ldots, h_0^s, h_1^s$ be chosen uniformly from $\{0,1\}^n$. Then, the probability that there exists $b_1, \ldots, b_s \in \{0,1\}$ such that $\bigoplus_i h_{b_i}^i = h$ is at least $1 - 2^{n-s}$. Furthermore, the sequence $b_1, \ldots, b_s$ can be efficiently computed.*

Key Idea of the Forging Attack. Now we describe the main idea of our forging attack. Our attack fixes nonce and associated data and so we simply denote the tag $T(N, U, S)$ by $T(S)$. Suppose M^0 is an encryption query with $T^0 := T(S^0)$ as a tag where S^0 denotes the S-value for the message M^0. Suppose C_i^0 and C_i^1 are the two i^{th} double-block ciphertexts for two different queries, $1 \le i \le s$. By Fact 1, we can find $b_1, \ldots, b_s$ such that $\oplus_{i=1}^s (C_i^{b_i}[0] \oplus C_i^{b_i}[1]) = S^0$. So if we can choose messages such that $\oplus_{i=1}^s (P_i^{b_i}[0] \oplus P_i^{b_i}[1]) = 0^n$ happens with high probability then $(C_1^{b_1}, \ldots, C_s^{b_s}, T^0)$ is a valid tagged ciphertext. As poly-hash is linear, we can ensure that $\oplus_{i=1}^s (P_i^{b_i}[0] \oplus P_i^{b_i}[0]) = 0^n$ holds with high probability for suitably chosen queries.

Forging Algorithm $\mathcal{F}_0$. Now let us fix a positive integer ℓ whose exact value will be determined later. We define the following messages

$$M^i := ((0,0)^{i-1}, (0,1), (0,0)^{\ell-i}), \quad 1 \le i \le \ell.$$

Let M^0 be the all zero block message. Our forging algorithm makes $\ell + 1$ many queries, namely M^i's.

Forging Algorithm $\mathcal{F}_0$ for COBRA.

1. It makes encryption queries M^i and obtains responses (C^i, T^i), $0 \le i \le \ell$.
2. Let $C^0 = (C_1^0[1], C_1^0[2], \cdots, C_\ell^0[1], C_\ell^0[2])$ and $h_0^i = C_i^0[1] \oplus C_i^0[2]$.
3. For $i = 1$ to ℓ
   let $C^i = (C_1^i[1], C_1^i[2], \cdots, C_\ell^i[1], C_\ell^i[2])$ and $h_1^i = C_i^i[1] \oplus C_i^i[2]$.
4. Let $h = h_0^\ell \oplus (\bigoplus_{i=1}^{\ell-1} h_0^i)$ (the sum of the ciphertext blocks for M^0).
5. Based on Fact 1, it finds a sequence $b_1, \ldots, b_{\ell-1} \in \{0,1\}$, $\bigoplus_{i=1}^{\ell-1} h_{b_i}^i = h_1^\ell \oplus h$.
6. If there is no such sequence then it aborts else it proceeds.
7. If $b_1 \oplus \cdots \oplus b_{\ell-1} \ne 1$ then it aborts.
8. Else it makes the forgery $(C^* := (C_1^*, \ldots, C_\ell^*), T^0)$ where for all $1 \le i \le \ell - 1$

$$C_i^* = \begin{cases} C_i^i[1] \| C_i^i[2] & \text{if } b_i = 1, \\ C_i^0[1] \| C_i^0[2] & \text{if } b_i = 0. \end{cases}$$

and $C^*[\ell] = C_\ell^\ell[1] \| C_\ell^\ell[2]$.

Now we compute the success probability of the forging attack. The forging algorithm aborts in two cases. We show that the abort probabilities are small. Moreover, given that it does not abort, we also show that the forging attack works perfectly.

Theorem 1. *The forgery algorithm $\mathcal{F}_0$ has success probability at least $\frac{1}{2} \times (1 - 2^{-n})$ when we set $\ell = 2n$.*

Proof. In the ideal case, h_0^i and h_1^i are independently and (almost) uniformly drawn from $\{0,1\}^n$ as these are xor of two blocks of the i^{th} double-block ciphertext for fresh queries M^i and M^0 respectively. Note that M^i and M^0 have different double block values in the ith position. By Fact 1, with probability at least $1/2$, we can efficiently find $b_1, \ldots, b_{\ell-1}$ such that $\oplus_{j=1}^{\ell-1} h_{b_j}^j = h \oplus h_1^\ell$.

Claim. Let us assume that we have found such $b_1, \ldots, b_{\ell-1} \in \{0,1\}$ which happens with probability at least $1 - 2^{n-\ell}$. Then,

$$b_1 \oplus \cdots \oplus b_{\ell-1} = 1 \Rightarrow (C^*, T) \text{ is a valid ciphertext tag pair.}$$

We first note that (C^*, T) is a fresh tagged ciphertext as the last double block of ciphertext is different from those of all other tagged ciphertexts. To prove that tagged ciphertext is valid, we first compute S^* and S^0 for the given forged ciphertext and M^0 respectively where S^0 denotes the S values for the message M^0.

Computation of S^0. Computation of S^0 is straightforward from its definition.

$$S^0 := (\oplus_{j=1}^{\ell}(P_j^0[1] \oplus P_j^0[2])) \oplus (\oplus_{i=1}^{\ell}(C_i^0[1] \oplus C_i^0[2]).$$

Now note that $P_i^0[1] = \mathsf{poly}_L(1, N, 0^{2i-2}, 0)$ and $P_0^i[2] = \mathsf{poly}_L(1, N, 0^{2i-1}, 0)$. Let $\Sigma = \bigoplus_{i=1}^{\ell}(\mathsf{poly}_L(1, N, 0^{2i-1}, 0) \oplus \mathsf{poly}_L(1, N, 0^{2i-2}, 0)$. So

$$S^0 = h \oplus (\bigoplus_{i=1}^{\ell}(\mathsf{poly}_L(1, N, 0^{2i-1}, 0) \oplus \mathsf{poly}_L(1, N, 0^{2i-2}, 0)) = h \oplus \Sigma.$$

Computation of S^*. Now we compute S^* under the assumption that the first abort does not hold, i.e., we have found $b_1, \ldots, b_{\ell-1}$ such that $\oplus_{j=1}^{\ell-1} h_{b_j}^j = h \oplus h_1^\ell$. Note that the xor of ciphertext blocks which is equal to $\oplus_{j=1}^{\ell-1} h_{b_j}^j \oplus h_1^\ell = h$.

Now we decrypt the forged ciphertext double blocks by applying $\mathsf{2LR}^{-1}$. Let $P_i^* := (P_i^*[1], P_i^*[2])$ be the i^{th} double block of forged ciphertext after we apply Luby-Rackoff two round decryption. Similarly, we denote P_i values for M^j query as P_i^j. As all ciphertext double blocks C_i^* have appeared in responses of queries at the same position, the P_i^* values, $1 \le i \le \ell$ are given as below.

$$P_i^* = \begin{cases} P_i^i[1] \| P_i^i[2] & \text{if } b_i = 1, \\ P_i^0[1] \| P_i^0[2] & \text{if } b_i = 0. \end{cases}$$

and $P^*_\ell = P^\ell_\ell[1] \| P^\ell_\ell[2]$. Note that

1. $P^i_i[1] = \mathsf{poly}_L(1, N, 0^{2i-1})$ and $P^i_i[2] = \mathsf{poly}_L(1, N, 0^{2i-1}1)$,
2. $P^0_i[1] = \mathsf{poly}_L(1, N, 0^{2i-1})$ and $P^0_i[2] = \mathsf{poly}_L(1, N, 0^{2i})$.

By linearity of poly_L, we can simply write for $1 \le i \le \ell - 1$,

1. $P^*_i[1] \oplus P^*_i[2] = \mathsf{poly}_L(1, N, 0^{2i-1}) \oplus \mathsf{poly}_L(1, N, 0^{2i}) \oplus b_i$ and
2. $P^*_\ell[1] \oplus P^*_\ell[2] = \mathsf{poly}_L(1, N, 0^{2\ell-1}) \oplus \mathsf{poly}_L(1, N, 0^{2\ell}) \oplus 1$.

So $\bigoplus_{j=1}^{\ell}(P^*_j[1] \oplus P^*_j[2]) = \Sigma \oplus 1 \oplus (\oplus_{j=1}^{\ell-1} b_j)$ and

$$S^* = \oplus(\Sigma \oplus 1 \oplus (\oplus_{j=1}^{\ell-1} b_j).$$

Now if the second abort does not hold then (i.e., $\oplus_j b_j = 1$) we have $S^* = h \oplus \Sigma = S^0$. This proves the claim.

Probabilities of the Abort Events. Now we informally argue that the second abort probability is $\mathsf{Pr}[\oplus_{j=1}^{\ell-1} b_j = 1] = 1/2$. Note that $C^i_i[1]$, $C^i_i[2]$, $C^0_i[1]$, $C^0_i[2]$'s are independent and so are h^i_0, h^i_1 for all $1 \le i \le \ell$. Thus by conditioning h^ℓ_0, h^ℓ_1, choices of b_i's are independent and uniform. So the probability is $1/2$. By Fact 1, the first abort does not hold with probability $1 - 2^{n-\ell}$ and now we claim the second abort does not hold with probability $1/2$. Hence success probability of forging is at least $\frac{1}{2}(1 - 2^{n-\ell})$ which is almost $1/2$ if we set $\ell = 2n$. □

Remark 1. Note that in the above attack, we can verify whether the forged ciphertext tag pair is valid without querying it. So we can repeat this process n times (we choose bit 0 or 1 in different position instead of the last bit as described above) to succeed with probability of about $1 - 2^{-n}$.

Remark 2. In the above analysis we make several probabilistic assumptions to make the analysis clean and simple. Here we list these.

1. We assume that h^i_j's are independent and uniform. However, for a fixed i, h^i_1 and h^i_0 are not completely independent as these are generated from ideal online random permutation. However, for these $4n$ outputs $C^i_i[1], C^i_i[2]$, $C^0_i[1], C^0_i[2]$ these are statistically very close to the uniform distribution with distance about $\binom{4n}{2}/2^n$.
2. True distributions of b_i's may not be uniform and independent. It actually depends on how we define b_i's as there could be more than one choice of b_i's. However, all of these choices would lead to abort with probability of about $1/2$ or less.

5 Security Analysis of POET and POET-m

In this section we analyze POET and its parallel variant POET-m for some positive integer m.

Algorithm: POET-m Encryption
Input: $(M_1, M_2, ..., M_\ell) \in (\{0,1\}^n)^\ell$
Output: $(C_1, C_2, ..., C_\ell, T) \in (\{0,1\}^n)^{\ell+1}$

---

for $i = 1$ to $\ell - 1$
  $X_i = \tau \oplus F_{L^{top}}(M_1 \oplus L_1) \oplus F_{L^{top}}(M_2 \oplus L_2) \oplus \cdots \oplus F_{L^{top}}(M_i \oplus L_i)$.
  $Y_i = E_K(X_i)$;
  $C_i = F_{L^{bot}}(Y_{i-1} \oplus Y_i) \oplus L_i$;
end for loop
$X_\ell = F_{L^{top}}(X_{\ell-1}) \oplus M_\ell$.
$Y_\ell = E_K(X_\ell)$;
$C_\ell = F_{L^{bot}}(Y_{\ell-1} \oplus Y_\ell)$;
$X_{\ell+1} = F_{L^{top}}(X_\ell) \oplus S \oplus \tau$.
$Y_{\ell+1} = E_K(X_{\ell+1})$;
$T = F_{L^{bot}}(Y_\ell) \oplus Y_{\ell+1} \oplus S$;
Return $(C_1, C_2, ..., C_\ell, T)$

Algorithm 2: POET-m encryption algorithm for a messages M of sizes ℓn with $\ell < m$. Let τ be an n-bit element which is derived from associated data. The elements $L_1, \ldots, L_{m-1}$ are derived keys and S is a key derived from length of the message.

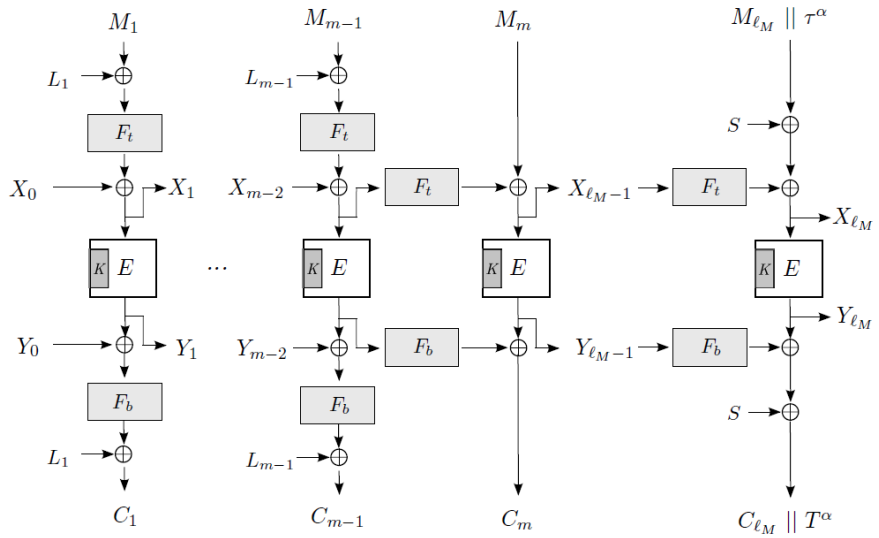

Fig. 5.1. POET-m Mode for ciphertext and tag generation. $X_0 = Y_0 = \tau$ is obtained from the associated data. We denote $F_{L^{top}}$ and $F_{L^{bot}}$ simply by F_t and F_b respectively.

5.1 POET-m and Its Security Analysis.

POET-m: We first describe ciphertext generation algorithm of parallel version POET-m. We consider F_L to be the field multiplier hash in which message block is multiplied by the key L. We describe how POET-m works for all messages $(M_1, \ldots, M_\ell)$ with $\ell < m$. Let τ be an n-bit element which is derived from associated data. The elements $L_1, \ldots, L_{m-1}$ are keys derived by invoking pseudorandom permutation on different constants (see [1, 3] for details). Note that the input of the blockcipher X_i is a sum hash. When we instantiate the AXU by field multiplier we can simplify the sum hash (due to linearity). We have

$$X_i = \tau \oplus L^{top} \cdot (M_1 \oplus \cdots \oplus M_i) \oplus L'$$

where L' is the remaining part depending only on keys. We use this expression to mount the attack.

Privacy Attack on POET-m. We first demonstrate a distinguishing attack on POET-m distinguishing it from uniform random online cipher when $m > 4$. We make two queries

1. $M = (M_1, M_2, M_3, M_4)$ and
2. $M' = (M'_1, M'_2, M'_3 := M_3, M'_4)$ such that $M_1 \neq M'_1$ and $M_1 \oplus M_2 = M'_1 \oplus M'_2$.

We denote the corresponding internal variables by X, C's and X', C''s. It is easy to see that $X_2 = X'_2$ and $X_3 = X'_3$ and hence $C_3 = C'_3$ with probability one. This equality of third ciphertext block happens with probability 2^{-n} for uniform random online cipher. So we have a distinguisher which succeeds with probability almost one. The presence of fourth block makes sure that X_i's are defined as above (as the final block is processed differently). We can keep all other inputs, for example nonce, associated data etc., the same.

Forging Attack on POET-m. Now we see how we can exploit the above weakness in sum of AXU hash to mount a forging attack on the construction. We can forge when the number of message blocks is less than m and the last block is complete (as described in Algorithm 2). We first simply describe how the decryption algorithm works. Assume $m > 3$ and let C_1, C_2, C_3, T be an input for decryption where $C_i, T \in \{0,1\}^n$. We note the following observations:

1. Y_i depends on $C_1 \oplus \cdots \oplus C_i$ for $i \leq 3$.
2. Verification algorithm depends on X_3, Y_3, T and some fixed values depending on associated data and key.

We make one query $M = (M_1, M_2, M_3)$ and obtain the response (C, T) where $C = (C_1, C_2, C_2)$. Let $C' = (C'_1, C'_2, C'_3 := C_3) \neq C$ such that $C_1 \oplus C_2 = C'_1 \oplus C'_2$. We denote the corresponding internal variable by X, C's and X', C''s. It is easy to see that by choices of C' and the first observation $Y_3 = Y'_3$ and so $X_3 = X'_3$. Again by the second observation, we see that verification algorithm depends on X_3, Y_3, T (or X'_3, Y'_3, T') and some fixed information based on key and associated

data. So, whenever verification algorithm passes for X_3, Y_3, it must pass for X'_3, Y'_3. Thus, (C'_1, C'_2, C_3, T) is a valid forge.

Note that the above attack is a single query forging attack and hence it is also applicable to situations where nonce can not be reused.

5.2 Security Analysis of POET mode

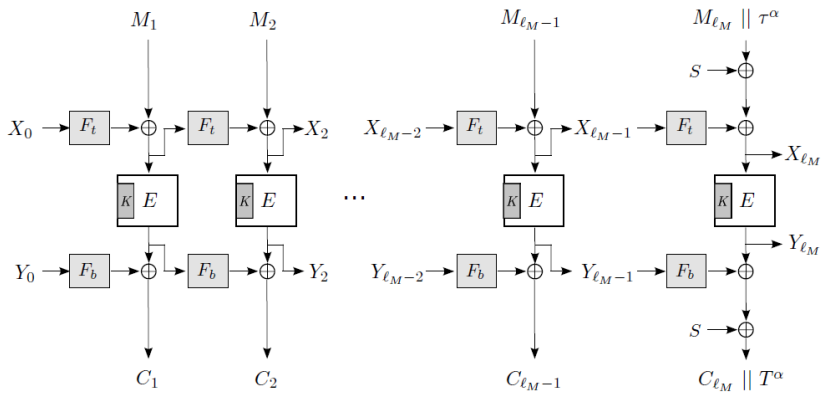

Fig. 5.2. POET Mode for ciphertext and tag generation. $X_0 = Y_0 = \tau$ is obtained from the associated data. In this figure, let F_t and F_b any independent ϵ-AXU hash functions.

POET: We now describe ciphertext generation algorithm of POET, i.e. POE the underlying encryption algorithm. In [1], the following theorem (restated) was claimed:

Theorem 8.1 of POET submission in [1]. Let E be a uniform random permutation, F_t and F_b be independent ϵ-AXU hash functions. Then, for any privacy adversary A making at most q queries of a total length of at most σ blocks, we have

$$\mathbf{Adv}^{\text{priv}}_{\mathsf{POET}^E}(A) \leq \epsilon\sigma^2 + \frac{\sigma^2}{2^n - \sigma}.$$

Here we consider F_t and F_b to be any arbitrary AXU functions as mentioned above in Theorem 8.1 of the submission POET in [1]. Given messages $(M_1, \ldots, M_\ell)$, we compute for $1 \leq i \leq \ell - 1$ as follows:

$$C_i = F_b(Y_{i-1}) \oplus Y_i,\ Y_i = E_K(X_i),\ X_i = F_t(X_{i-1}) \oplus M_i$$

where $X_0 = Y_0 = \tau$. The last ciphertext block is computed differently and we do not need its description for our distinguishing attack. Note that X_i is computed

by CBC^{F_t}. If F_t is a uniform random involution then as we have seen before, after applying CBC mode to F it does not remain δ-AXU for all $\delta < 1$. We use this property to obtain a distinguisher.

Privacy Attack on POET with uniform random involution F_t. Now we demonstrate a privacy attack on POET distinguishing it from uniform random cipher when F_L is instantiated with uniform random involution. In this attack we only make a single query and so it is also nonce-respecting. This would violate Theorem 8.1 of the submission POET in [1] (online permutation security of POE). We believe that the theorem remains valid when F_L is instantiated with field multiplier (however, proof needs to be revised). The attack is described below.

Claim. $\mathsf{Pr}[C_2 = C_4] = 1$ where $(C_1, C_2, \ldots)$ is the response of $(M_1, 0, 0, 0, \cdots)$ to POET with involution F_t.

We prove the claim by using the involution property of F. We can easily see that

1. $X_3 = F(F(X_1)) = X_1$ and
2. similarly, $X_4 = X_2 = F(X_1)$.

So $Y_1 = Y_3$ and $Y_2 = Y_4$ and hence $C_2 = C_4$. Note, we can choose any arbitrary nonce and associated data. This proves the claim.

In an ideal case, we observe $C_2 = C_4$ with probability 2^{-n}. So the distinguisher of POET has advantage at least $1 - 2^{-n}$.

6 Conclusion

In this paper, we demonstrate forging attack on COBRA with practical complexity. Hence the theorem proved in [4] is wrong. We also demonstrate forging and distinguishing attack on POET-m for one particular recommended choice of AXU hash function. We also disprove the security claim for POET by presenting a distinguishing attack on a different choice of AXU hash function (not in the recommended list). However, these attacks on POET do not carry over to the versions submitted to CAESAR.

Acknowledgement. This work is supported by Centre of Excellence in Cryptology at Indian Statistical Institute, Kolkata. Author would also like to thank all anonymous reviewers who provided us very useful comments to improve the quality of the paper.

References

1. CAESAR submissions, 2014. http://competitions.cr.yp.to/caesar-submissions.html.

2. ISO/IEC 9797. Data cryptographic techniques-Data integrity mechanism using a cryptographic check function employing a blockcipher algorithm, 1989.
3. Farzaneh Abed, Scott Fluhrer, Christian Forler, Eik List, Stefan Lucks, David McGrew, and Jakob Wenzel. Pipelineable on-line encryption. *Fast Software Encryption, LNCS. Springer, to appear*, 3:320–337.
4. Elena Andreeva, Atul Luykx, Bart Mennink, and Kan Yasuda. Cobra: A parallelizable authenticated online cipher without block cipher inverse. *Fast Software Encryption, LNCS. Springer, to appear*, 2014.
5. Mihir Bellare. New proofs for nmac and hmac: Security without collision-resistance. *IACR Cryptology ePrint Archive*, 2006:43, 2006.
6. Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of cipher block chaining. In Yvo Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 341–358. Springer, 1994.
7. Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *Advances in CryptologyEUROCRYPT97*, pages 163–192. Springer, 1997.
8. John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. UMAC: Fast and Secure Message Authentication. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 216–233. Springer, 1999.
9. Larry Carter and Mark N. Wegman. Universal Classes of Hash Functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.
10. Joan Daemen, Mario Lamberger, Norbert Pramstaller, Vincent Rijmen, and Frederik Vercauteren. Computational aspects of the expected differential probability of 4-round aes and aes-like ciphers. *Computing*, 85(1-2):85–104, 2009.
11. Joan Daemen and Vincent Rijmen. The Design of Rijndael: AES - The Advanced Encryption Standard., 2002. http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf.
12. Edgar N Gilbert, F Jessie MacWilliams, and Neil JA Sloane. Codes which detect deception. *Bell System Technical Journal*, 53(3):405–424, 1974.
13. Jian Guo, Jérémy Jean, Thomas Peyrin, and Lei Wang. Breaking poet authentication with a single query. Technical report, Cryptology ePrint Archive, Report 2014/197, 2014. http://eprint. iacr. org.
14. W. G. Horner. *Philosophical Transactions, Royal Society of London*, 109:308–335, 1819.
15. M. Luby and C. Rackoff. How to construct pseudo-random permutations from pseudo-random functions. In *Advances in Cryptology – Crypto 1985*, number 218 in Lecture Notes in Computer Science, page 447, New York, 1984. Springer-Verlag.
16. Mridul Nandi. The characterization of luby-rackoff and its optimum single-key variants. In Guang Gong and Kishan Chand Gupta, editors, *INDOCRYPT*, volume 6498 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2010.
17. Noam Nisan and David Zuckerman. Randomness is linear in space. *J. Comput. Syst. Sci.*, 52(1):43–52, 1996.
18. P. Rogaway. Bucket hashing and its application to fast message authentication. In *Advances in Cryptology – Crypto 1995*, number 963 in Lecture Notes in Computer Science, pages 29–42, New York, 1995. Springer-Verlag.
19. Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes ocb and pmac. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

20. Victor Shoup. On fast and provably secure message authentication based on universal hashing. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 313–328. Springer, 1996.
21. D. R. Stinson. On the connections between universal hashing, combinatorial designs and error-correcting codes. In *Congressus Numerantium*, number 114, pages 7–27, 1996.
22. Douglas R. Stinson. Universal hashing and authentication codes. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 74–85. Springer, 1991.
23. Douglas R. Stinson. Universal hashing and authentication codes. *Des. Codes Cryptography*, 4(4):369–380, 1994.
24. Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981.