Forging Attacks on two Authenticated Encryptions 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 in 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 about 1/2. This disproves the claim proved in FSE 2014 paper. We also show both privacy and forging attack on the parallel version of POET, denoted POET-m. We can also recover some derived key of the construction. In case of the modes POET or POE (the underlying modes for encryption), we show one query distinguishing attack when we instantiate the underlying AXU-hash function with some other AXU hash function, namely uniform random involution. Thus, our result violates the designer's main claim (Theorem 8.1 in [1]). However, the attacks can not be extended directly for the specific choices of existing submitted versions to the CAESAR competition.

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

1 Introduction

The common application of cryptography is to implement a secure channel between two or more users and then exchanging 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* Prf and (tweakable symmetric-key) *encryption* $\mathsf{Enc}(\cdot)$ respectively.

- The encryption Enc provides **privacy** or **confidentiality** *plaintext* or *payload* 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* $A \in \mathcal{D}$). Mac also provides **user-authenticity** (protecting from impersonation).

An **Authenticated Encryption** scheme (or simply **AE**) serves the both purposes in an integrated manner. An authenticated encryption scheme AE has two functionalities one of which, called tagged-encryption, essentially combines tag-generation and encryption, and the other combines verification and decryption algorithms.

1. **Tagged-encryption** $\mathsf{AE}.\mathrm{enc}_k$: It takes a message M from its message space $\mathcal{M} \subseteq \{0,1\}^{\leq L_{\mathrm{mm}}} := \cup_{i=1}^{L_{\mathrm{mm}}}\{0,1\}^i$ (L_{mm} denotes the maximum possible message size) and an associated data $A \in \mathcal{D}$ as inputs and generates a ciphertext integrated with tag $Z = (C,T) \in \{0,1\}^*$, called **tagged-ciphertext**.
2. **Verified-decryption** $\mathsf{AE}.\mathrm{dec}_k$: It takes a tagged ciphertext Z and an associated data A as inputs and returns either a special symbol $\perp$ (meaning that the given tagged-ciphertext is rejected i.e., *invalid*) or it returns a plaintext M when Z is actually the tagged-ciphertext of M with the associate data A.

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 usually includes keys for underlying blockcipher, masking keys etc. Some constructions derives more keys by invoking the blockcipher with different constant inputs.

An AE scheme is said to have *privacy* if the tagged ciphertext for any plaintext chosen adaptively behave like an uniform random string. It has *authenticity* if it is infeasible to generate a new valid ciphertext-tag pair which is not obtained before by making encryption query. More formally, let A be an oracle adversary which can make query to $\mathsf{AE}.\mathrm{enc}$ adaptively. Let \$ be a random oracle which returns an uniform random string for every fresh query. We define privacy advantage of A against AE to be

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

Similarly we define authenticity advantage of A as $\mathbf{Adv}^{\mathrm{auth}}_{\mathsf{AE}}(A) := \mathsf{Pr}[A^{\mathsf{AE}.\mathrm{enc}} = Z]$ where Z is valid tagged-ciphertext not a response of its encryption query.

1.1 Two AE Schemes COBRA and POET submitted to CAESAR

CAESAR [1], a competition for authenticated encryption having security applicability, and robustness. The final goal of the competition is to identify a portfolio of AE depending on different applications and environments. Fifty seven authenticated encryptions have been submitted. AES-COBRA and POET are two such submissions. These two have been published before in FSE 2014. Unfortunately, in [13] Guo et al. demonstrated one query forging attack of POET. So designers of POET modified accordingly to resist this forging attack and submitted the revised version to CAESAR.

1.2 Our Contribution

1. In this paper we show 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 $2n$ many encryption queries where n is the plaintext size of the underlying blockcipher. Our technique seems to be applicable to all stateless authenticated encryptions which follow hash then encrypt paradigm.

2. The designers of POET has recommended a parallel version, called POET-m. We provide distinguishing and forging attack on it. Moreover designers of POET claimed AE security of the encryption mode, called POE, for any arbitrary AXU hash function. Here we disprove their claim by showing a distinguishing attack on a special choice of AXU, namely uniform involution function. Thus the security proof of the claims have flaws. We also extended this to have a forging attack. All these attack algorithms make at most encryption queries and has advantages close to 1. However, we would like to note that our attacks on POET mode do not work for the choices of AXU which are proposed in CAESAR submissions.

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 blockciphers. We mostly use AES (advanced encryption standard) [11] with 128 bit key size as the underlying blockcipher and in this case $n = 128$. For any set S, we write $S^+ := \cup_{i \geq 1} S^i$.

Binary Field. We identify $\{0,1\}^n$ 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 an $(n-1)$ degree polynomial with coefficient $\alpha_0, \alpha_1, ..., \alpha_{n-1}$, i.e, $\alpha(x) = \alpha_0 + \alpha_1 x + \cdots + \alpha_{n-1}x^{n-1}$. 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, $0 = 0^n$ is the additive identity and $1 = 0^{n-1}1$ is the multiplicative identity. Moreover, $2 = 0^{n-2}10$ is a primitive element.

2.2 Almost XOR Universal (AXU) Hash

Universal hash functions and its close variants *strongly universal*, *AXU-hash* [9, 12, 21, 23, 24, 22, 18] are information theoretic notion which are used as building blocks of several cryptographic construction, e.g., *message authentication code* [9, 25], domain extension of pseudorandom function [5, 8], extractor [17], quasi-randomness and other combinatorial objects [12, 22].

AXU Hash Function. A keyed function $F_L : \{0,1\}^n \to \{0,1\}^n$ is called ϵ-AXU [18] if for all $x \neq x' \in \{0,1\}^n$ and $\delta \in \{0,1\}^n$, $\mathsf{Pr}_L[F_L(x) \oplus F_L(x') = \delta] \leq \epsilon$.

2.3 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 [21] 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 1. *[21] We define the polynomial-hash indexed by $L \in \{0,1\}^n$ over the domain $(\{0,1\}^n)^+$ as*

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

where $a_0, a_1, \ldots, a_d \in \{0,1\}^n$.

Four Rounds AES. The AES (for 128 bit keys) has ten rounds. However, it has been studied that the four rounds of AES has good differential probability. More formally, Daemen et al. in [10] showed that the four-round AES is a family of 2^{-113}-AXU under the reasonable assumption that all four round keys are uniform and independent (which is actually not the case for AES).

Random (Involution) Function. 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 involution if f is inverse of itself (so it must be permutation). An uniform random involution function I_n is chosen uniformly from the set $\mathcal{I}_n$ of all involution functions from $\{0,1\}^n$ to itself. It is easy to see that I_n is an $\frac{1}{2^n-2}$-AXU hash function.

2.4 Combination of AXU hash functions

Compositions of AXU hash functions Now we show that AXU-hash function does not preserve under composition with same key. In other words, if F_L is ϵ-AXU then $F_L \circ F_L$ is not necessarily AXU. It can be easily seen for uniform random involution function as the composition is identity function which is clearly not an AXU hash function. Similar result holds if we apply CBC composition of F. That is, for any positive integer $CBC^{F_L} : (\{0,1\}^n)^\ell \to \{0,1\}^n$ is not AXU. Note that

$$CBC^f(x_1, \ldots, x_\ell) = y_\ell, \text{ where } y_i = f(y_{i-1} \oplus x_i), 1 \le i \le \ell$$

and $y_0 = 0^n$. So $CBC^{F_L}(x_1, 0) = F_L(F_L(x_1))$ which is not necessarily AXU hash function as we have seen before. However, it is true for specific choice, e.g. when F_L is field multiplier. In this case, CBC^{F_L} is nothing but poly-hash which is $\ell 2^{-n}$-AXU for fixed ℓ. Even for variable ℓ with appropriate padding on message, we can make it AXU hash function.

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 $F_L^{sum}(x_1, \ldots, x_\ell) = F_L(x_1) \oplus \cdots \oplus F_L(x_\ell)$. Note that if F_L is linear (which is true for field multiplier) then the sum hash is clearly not AXU. However, the sum hash is AXU when we consider uniform random (involution) functions and we apply counter on message blocks. More precisely, $F_L(x_1\|1) \oplus \cdots \oplus F_L(x_\ell\|\ell)$ is $\frac{1}{2^n-2\ell}$-AXU. Instead of counter, one can apply masking by keys indexed by position which can be derived from the random function.

3 Description of COBRA

COBRA is an authenticated encryption mode based on blockcipher. It is originally published in FSE 2014 [4]. Later the same mode with AES as the underlying blockcipher, called AES-COBRA, submitted to CAESAR [1]. The mode can be viewed as hash then ECB type where hash function is poly-hash type and ECB is applied on a double block (i.e., $2n$ bit plaintext) encryption which is defined by two rounds Fiestel structure [15]. As it uses Fiestel structure, it is an inverse-free. In other words, even though it is based on AES blokcipher, the decryption of COBRA does not require AES decryption. Other than implementation advantage, it has potentially advantage in security since we only need to rely on the pseudorandom permutation assumption of AES instead of strong pseudorandom permutation.

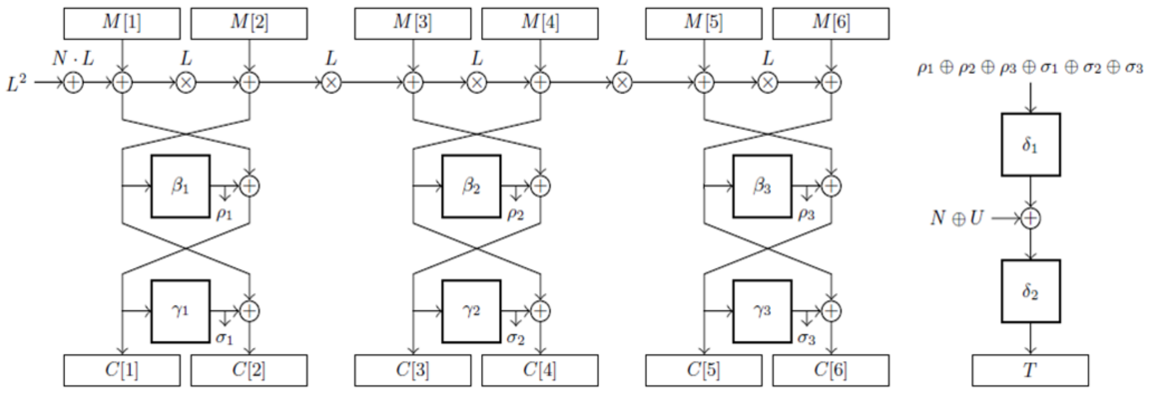

Fig. 3.1. COBRA Modes for ciphertext and tag generation for three double blocks message. U is obtained from associated data and L is the hash key.

COBRA defines for any messages of size at least n bits and it expands n bits in its corresponding ciphertext. Now we briefly describe how the encryption algorithm of COBRA works for all inputs $M \in (\{0,1\}^{2n})^+$. In addition with 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 sizes inputs). We write $M = M_1 \| \cdots \| M_\ell$ for some positive integer ℓ where $M_1, \ldots, M_\ell \in \{0,1\}^{2n}$. We also write $M_i = (M_i[1], M_i[2])$ where $M_i[2], M_i[2] \in \{0,1\}^n$ 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) permutation over $\{0,1\}^n$ for all $i \geq 1$. We describe the COBRA-mode based on these permutations. These are actually derived from a single blockcipher using the standard masking algorithm (i.e., XEX construction [19]). For details description of tweakable pseudorandom permutations readers can see [4, 1]).

Algorithm: COBRA Encryption
Input: $(M_1, M_2, ..., M_d) \in (\{0,1\}^{2n})^d$, $N \in \{0,1\}^n$
Output: $(C_1, C_2, ..., C_d) \in (\{0,1\}^{2n})^d$

```
1  for $i = 1$ to $d$
2      write $M_i = M_i[1] \| M_i[2]$, $M_i[1], M_i[2] \in \{0,1\}^n$
3      $P_i[1] = \mathsf{poly}_L(1, N, M_1[1], M_1[2], \ldots, M_i[1])$;
4      $P_i[2] = \mathsf{poly}_L(1, N, M_1[1], M_1[2], \ldots, M_i[1], M_i[2])$;
5      $C_i[1] = P_i[1] \oplus \beta_i(P_i[2])$;
6      $C_i[2] = P_i[2] \oplus \gamma_i(C_i[2])$;
7      write $C_i = C_i[1] \| C_i[2]$;
8  end for loop
9  Return $(C_1, C_2, ..., C_d)$
```

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

The line 3 and 4 of the above algorithm in which ciphertext are computed can be viewed as a 2-round Feistel structure.[1] We denote this computation as

$$\mathsf{2LR}^{\beta_i,\gamma_i}(P_i[1], P_i[2]) = (C_i[1], C_i[2]).$$

We also simply write the above as $\mathsf{2LR}_i(\cdot,\cdot)$. It is easy to see that it is invertible and the inverse function $\mathsf{2LR}_i^{-1}(C_i[1], C_i[2]) = (P_i[1], P_i[2])$ where $P_i[2] = \gamma_i(C_i[2]) \oplus C_i[1]$ and $P_i[1] = \beta_i(P_i[2]) \oplus C_i[1]$. The COBRA mode is categorized as inverse-free as the decryption algorithm does not require any inverse of tweakable random permutation. So one can replace the tweakable random permutations by tweakable random functions.

3.1 Tag Generation

The final tag $T(S, N, U)$ is computed from

$$S := \oplus_i (P_i[1] \oplus P_i[2] \oplus C_i[1] \oplus C_i[2]),$$

nonce N and U which depends only on the associated data A. One can find the details of the construction in [4, 1]. Now we state a simple but important observation which would be used to analyze our forging attack on this mode.

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

Lemma 1. *If for two distinct triples (A, N, M) and (A', N', M') such that $\mathsf{Pr}[S = S'] = p$, $N = N'$, and $U = U'$ (which is true if we take $A = A'$) then the probability that their tag matches is p:*

$$\mathsf{Pr}[T(S, N, U) = T(S', N', U')] = p.$$

In our forging attack we keep nonce N and associated data A same and so it would be sufficient to find $M \neq M'$ such that its corresponding S and S' match with high probability. This would lead to forge a tag ciphertext pair. As our attack fixes nonce and associated data we denote the tag simply by $T(S)$.

4 Forging Attack on COBRA

Let us fix an integer ℓ. Later we see the choice of it. We define the following messages

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

Let M^0 be the all zero block message. Now we briefly describe the forging algorithm.

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

```
1. Make encryption queries $M^i$ and obtains responses $(C^i, T^i)$, $0 \leq i \leq \ell$.
2. Parse $C^0 = (C_1^0[1], C_1^0[2], \cdots, C_\ell^0[1], C_\ell^0[2])$.
3. For $i = 1$ to $\ell$
   (a) Parse $C^i = (C_1^i[1], C_1^i[2], \cdots, C_\ell^i[1], C_\ell^i[2])$,
   (b) Let $h_1^i = C_i^i[1] \oplus C_i^i[2]$.
   (c) Let $h_0^i = C_i^0[1] \oplus C_i^0[2]$.
4. Let $h = h_0^\ell \oplus (\bigoplus_{i=1}^{\ell-1} h_0^i)$.
5. Find 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 abort else we proceed.
7. If $b_1 \oplus \cdots \oplus b_{\ell-1} \neq 1$ then abort.
8. Else make the forgery $(C^* := (C_1^*, \ldots, C_\ell^*), T^0)$ where for all $1 \leq i \leq \ell - 1$
   $$C_i^* = \begin{cases} C_i^i[1] \| C_i^i[2] & \text{if } b_i = 1, \\ C_i^0[1] \| C_i^0[2] & \text{if } b_i = 0. \end{cases}$$
   and $C^*[\ell] = C_\ell^\ell[1] \| C_\ell^\ell[2]$.
```

The forging algorithm makes $\ell + 1$ many queries. This algorithm aborts in two cases. We need to compute the abort probabilities. Given that it does not abort we also have to show that the forging attack works. To compute the probability of the first abort, we apply the following fact.

Fact 1.[7] *Let $h \in \{0,1\}^n$ be a fixed element and $h_0^1, h_1^1, \ldots, h_0^\ell, h_1^\ell$ be chosen uniformly from $\{0,1\}^n$. Then, the probability that there exists $j_1, \ldots, j_\ell \in \{0,1\}$ such that $\bigoplus_j h_{j_i}^i = h$ is at least $1 - 2^{n-\ell}$.*

Theorem 1. *The forgery algorithm $\mathcal{F}_0$ has success probability at least $1/4$.*

Proof. For a random online cipher h_0^i and h_1^i will be independently drawn from $\{0,1\}^n$ as these are xor of two blocks of the i^{th} double-block ciphertext for M^i and M^0 queries respectively. Note that M^i and M^0 have different double block in ith position. By the above fact, with probability at least $1/2$, we have $b_0, \ldots, b_{\ell-1}$ such that $\oplus_{j=0}^{\ell-1} h_{b_j}^j = h \oplus h_1^\ell$.

Claim. Let us assume that we have such $b_0, \ldots, b_{\ell-1} \in \{0,1\}$ which can happen with probability at least $1 - 2^{n-\ell}$. Then,

$$b_0 \oplus \cdots b_{\ell-1} = 1 \Rightarrow (C^*, T) \text{ is a valid ciphertext tag pair.}$$

To prove the above claim we compute S^* and S^0 for the given forged ciphertext and M^0 respectively where S^i denotes the S values for the message M^i.

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

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

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

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

where Σ is defined to be the xor of the poly-hash values.

Computation of S^* Now we compute S^* under the assumption that the first abort does not hold. We first compute the xor of ciphertext blocks. As first abort does not hold, the xor of all forged ciphertext blocks is same as h.

Now we decrypt the forged ciphertext double blocks by $\mathsf{2LR}^{-1}$. 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 (i.e., after applying poly-hash in line 3 and 4). Similarly, we denote P_i values for M^j query as P_i^j. As all ciphertext double blocks C_i^{i,j_i} are appeared in responses of queries (keeping the position same) we can compute the P_i^* values easily for $1 \le i \le \ell - 1$ and these are

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

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

$$S^* = [\oplus_{j=0}^{\ell-1} h^j_{b_j} \oplus h^\ell_1] \oplus (\Sigma \oplus 1 \oplus (\oplus_{j=1}^{\ell-1} b_j).$$

By choices of $h^j_{b_j}$'s and if $\oplus_j b_j = 1$ implies that $S^* = h \oplus \Sigma = S^0$. This proves the claim.

Now we information argue that $\mathsf{Pr}[\oplus_{j=1}^{\ell-1} b_j = 1] = 1/2$. 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_0, choices of b_i's are independent and uniform. So the probability is 1/2. By the Fact 1, 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 $\ell = 2n$. Moreover, note that we can verify whether forged ciphertext tag pair is valid without querying it. So we can try encryption queries until we succeed. □

One can modify the above attack as follows. Whenever we abort (which can happen with probability about 1/2) we may repeat the process by choosing another set of M^i's in which choice of the bit 0 or 1 in different position instead of the last bit as described above. We can make n such tries and all these try fail with probability about 2^{-n}. So with probability very close to one we can forge. Note that we can decide whether abort events holds or not without making the forgery attempt.

Remark 1. In the above analysis we make several probabilistic assumptions to make it 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 uniform 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 close to 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 choices of b_i's. However all these choices would lead abort has probability about 1/2 or less.

5 Security Analysis of POET and POET-m

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

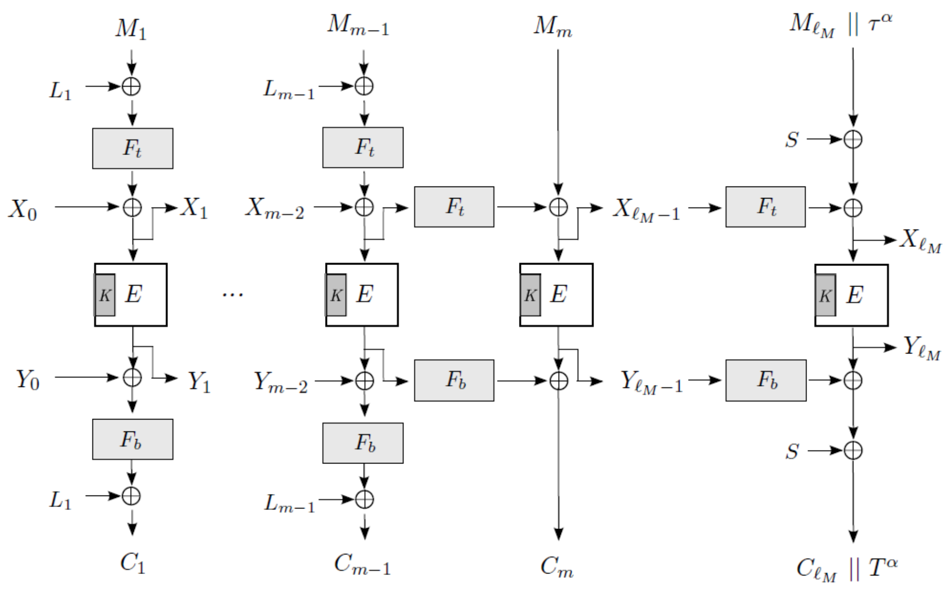

Fig. 5.1. POET-m Mode for ciphertext and tag generation. $X_0 = Y_0 = \tau$ which is obtained from the associated data.

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 encryption algorithm for a messages M of sizes ℓn with $\ell < m$. Let τ be an n bit elements 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.

L (example 1). We describe how POET-m works for all messages $(M_1, \ldots, M_\ell)$ with $\ell < m$. Let τ be an n bit elements which is derived from associated data. The elements $L_1, \ldots, L_{m-1}$ are derived keys 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.

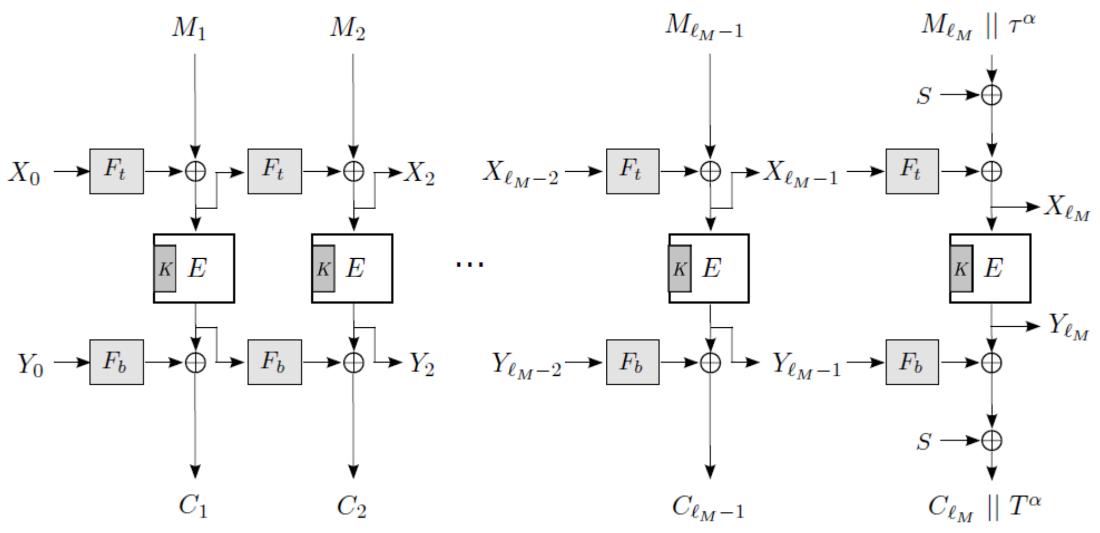

Fig. 5.2. POET-m Mode for ciphertext and tag generation. $X_0 = Y_0 = \tau$ which is obtained from the associated data.

POET: Now we describe only ciphertext generation algorithm of POET, i.e. POE the underlying encryption algorithm. Here we consider F and F' to be any arbitrary AXU functions (as mentioned 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:

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

where $X_0 = Y_0 = \tau$. The last ciphertext block is computed differently and we do not need the description for our distinguishing attack. Note that X_i is computed by CBC^F. If F is uniform random involution function then CBC does not remain AXU (it becomes poly-hash when F is field multiplier). We use this property to make a distinguisher.

5.1 Distinguishing Attack on POET-m and POET

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

$M = (M_1, M_2, M_3, M_4)$ and $M' = (M'_1, M'_2, M'_3, M'_4)$ such that $M_1 \neq M'_1$, $M_1 \oplus M_2 = M'_1 \oplus M'_2$ and $M_3 = M'_3$. So $X_2 = X'_2$ 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 almost one probability distinguisher. 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., same.

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

We make a single query with $(M_1, 0, 0, 0, \cdots)$. By involution property, we prove that $C_2 = C_4$ with probability one for POET. 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 $C_2 = C_4$. Again, we can choose any arbitrary nonce and associated data.

5.2 Key Recovery and Forging Attack on POET-m

Now we see how we can exploit the weakness in sum of AXU hash to actually a forge the construction. Here we first recover the key L_F which is used for AXU for both bottom and top layers by making only two queries.

5.3 Key Recovery of L_F

We make two queries as distinguishing attack $M = (M_1, M_2, M_3, M_4)$ and $M' = (M'_1, M'_2, M'_3, M'_4)$ such that $M_1 \neq M'_1$, $M_1 \oplus M_2 = M'_1 \oplus M'_2$ and $M_3 = M'_3$. So $X_2 = X'_2$ and $X_3 = X'_3$ and hence $C_3 = C'_3$. Now we observe that if $Y_1 \oplus Y_2 = \delta \neq 0$ then $C_1 \oplus C'_1 = L_F \cdot \delta$ (similarly for $C_2 \oplus C'_2$). So we obtain $L_F = \delta^{-1} \cdot (C_1 \oplus C'_1)$.

5.4 Forging attack on POET-m

Once we have L_F by making two encryption queries (with same nonce and associated data) the construction actually behaves like OCB [20] where the ith message block of OCB is actually some known linear function of $M_1, \ldots, M_i$ and similarly for the ith ciphertext. When nonce is reused, OCB can be forged very easily. We basically transform the message blocks corresponding to attack of OCB to that for POET-m. Similarly, when we forge we transform the ciphertext accordingly.

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 (through derived key-recovery) and distinguishing attack on POET-m for one particular recommended choice of AXU hash function. We also show the security claim for POET is wrong by showing a distinguishing attack on a different choice of AXU hash function (not in the recommended list). However, these attacks on POET does not carry through for the versions submitted to CAESAR.

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. Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, 2003.

21. 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.
22. D. R. Stinson. On the connections between universal hashing, combinatorial designs and error-correcting codes. In *Congressus Numerantium*, number 114, pages 7–27, 1996.
23. 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.
24. Douglas R. Stinson. Universal hashing and authentication codes. *Des. Codes Cryptography*, 4(4):369–380, 1994.
25. 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.