Tiaoxin − 346

VERSION 1.0

Codenames:
`tiaoxin, tiaoxinv1`

Designer/submitter:
Ivica Nikolić
Nanyang Technological University, Singapore
inikolic@ntu.edu.sg

15.03.2014

Chapter 1

Specification

1.1 Authenticated Encryption $\mathsf{Tiaoxin}-346$

$\mathsf{Tiaoxin}-346$ is a nonce-based authenticated encryption scheme.

In the encryption/authentication stage it takes four inputs:

- Key K of size 128 bits,
- Public message number (nonce) IV of size 128 bits,
- Plaintext (also called a message) M of size 0 to $2^{128}-1$ bits,
- Associated data AD of size 0 to $2^{128}-1$ bits,

and it outputs a ciphertext C and an authentication tag Tag, i.e.

$$\mathsf{Tiaoxin}-346\ (\mathsf{K}, \mathsf{IV}, \mathsf{M}, \mathsf{D}) = (\mathsf{C}, \mathsf{Tag}).$$

The number of bytes[1] of C equals the number of bytes of M. The length of the authentication tag Tag is at most 128 bits (inclusive). $\mathsf{Tiaoxin}-346$ does not use secret message number. The public message number is a nonce.

In the decryption/verification stage it takes key K, public message number (nonce) IV, ciphertext C, associated data AD, tag Tag and outputs fail if the tag is incorrect, or it outputs the message M if it is correct.

1.2 Recommended Parameters

$\mathsf{Tiaoxin}-346$ can parametrize the tag length. We recommend one parameter set:

1. Tag length of 128 bits. Codename: `tiaoxin-v1`.

[1] We assume the granularity is on a byte, rather than bit level. If required, the bit granularity can be applied easily.

1.3 Preliminaries

The following notations and operations are used in the paper.

Notations:

- Word - a sequence of 16 bytes. The values of words are given in hexadecimal notation. Word to AES matrix conversation is the standard one[2]: the first byte of the word (sequence) is at $(1,1)$ in the matrix, the fourth at $(4,1)$, the sixteenth is at $(4,4)$.
- Z_0 - a constant word defined as Z_0 =`428a2f98d728ae227137449123ef65cd`
- Z_1 - a constant word defined as Z_1 =`b5c0fbcfec4d3b2fe9b5dba58189dbbc`
- T_s - a state composed of s words. For instance, T_3 has 3 words, T_6 has 6 words. To index state words we use the language C notation, hence $T_s = (T_s[0], T_s[1], \dots, T_s[s-1])$, where $T_s[i], i = 0, \dots, s-1$ are words, and $T_s[0]$ is the first word.

Operations:

- $|X|$ - length of X expressed in bits
- $X||Y$ - concatenation of X and Y
- $X \oplus Y$ – bitwise addition (XOR) of the words X and Y
- $X \& Y$ – bitwise conjunction (AND) of the words X and Y
- $AES(X, SK)$ – one keyed round of AES applied to the word X, where SK is the subkey, i.e.:

  $$AES(X, SK) = MixColumns(ShiftRows(SubBytes(X))) \oplus SK.$$

  $SubBytes, ShiftRows, MixColumns$ are the same operations as in AES. Thus, $AES(X, SK)$ is the AES-NI instruction $aesenc$.
- $R(T_s, M)$ – a round transformation of a state with s words. The inputs of R are state T_s and word M, while the output is a new state T_s^{new}, i.e. $R : T_s \times M \rightarrow T_s^{new}$:

  $$\begin{aligned} T_s^{new}[0] &= AES(T_s[s-1], T_s[0]) \oplus M \\ T_s^{new}[1] &= AES(T_s[0], Z_0) \\ T_s^{new}[2] &= T_s[1] \\ &\dots \\ T_s^{new}[s-1] &= T_s[s-2] \end{aligned}$$

[2] This allows to use Intel's load/store and AES-NI instructions without any additional byte reversals, see [7].

1.4 The States of $\mathtt{Tiaoxin}-346$ and the $\mathtt{Update}$ Operation

$\mathtt{Tiaoxin}-346$ has three states T_3, T_4, T_6 composed of $3, 4, 6$ words, respectively. The $\mathtt{Update}$ operation (called a round function), based on the above $R(T_s, M)$, is used to compute the new value of the states (in the different phases). As inputs, beside the three states, $\mathtt{Update}$ takes three additional words M_0, M_1, M_2, i.e. $\mathtt{Update} : T_3 \times T_4 \times T_5 \times M_0 \times M_1 \times M_2 \rightarrow T_3 \times T_4 \times T_6$.

$\mathtt{Update}(T_3, T_4, T_5, M_0, M_1, M_2)$ is defined as (see Fig. 1.1):

$$T_3^{new} = R(T_3, M_0); T_3 = T_3^{new}$$
$$T_4^{new} = R(T_4, M_1); T_4 = T_4^{new}$$
$$T_6^{new} = R(T_6, M_2); T_6 = T_6^{new}$$

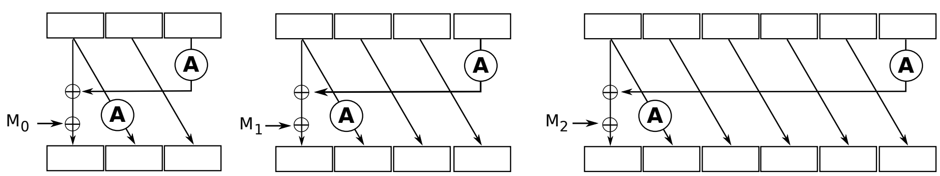

Figure 1.1: The $\mathtt{Update}$ operation (round function) in $\mathtt{Tiaoxin}-346$. Circled A stands for one AES round. The AES rounds applied to $T_3[2], T_4[3], T_6[5]$ are keyless, while the AES rounds applied to $T_3[0], T_4[0], T_6[0]$ use Z_0 as a subkey.

1.5 Definition of $\mathtt{Tiaoxin}-346$

Let us define the encryption-authentication step of the design. $\mathtt{Tiaoxin}-346$ processes the associated data AD and the message M in blocks where each block is composed of 2 words (32 bytes, 256 bits):

- The associated data AD is divided into blocks of 32 bytes each. If the last block of AD is incomplete (the length of the block is less than 32 bytes), this block is padded with zero bytes. The padded associated data is denoted as $\overline{AD}$ and hence $\overline{AD} = AD_1 \| AD_2 \| \ldots \| AD_d$, where $|AD_i| = 256$ and $d = \lceil \frac{|AD|}{256} \rceil$. If AD is empty, then $d = 0$, and $\overline{AD}$ is empty. The length of the AD is encoded as 16-byte big endian word and stored in AD_{lenght}, i.e. $AD_{lenght} = |AD|$.
- The message M is divided into blocks and padded with zero bytes if the last block has less than 32 bytes. The padded message is denoted as $\overline{M}$ and hence $\overline{M} = M_1 \| M_2 \| \ldots \| M_m$, where $|M_i| = 256$ and $m = \lceil \frac{|M|}{256} \rceil$. If

M is empty, then $m = 0$ and $\overline{M}$ is empty. The length of the M is encoded as 16-byte big endian word and stored in M_{lenght}, i.e. $M_{lenght} = |M|$.

$\mathit{Tiaoxin} - 346$ is a stream cipher based design and as such it works in four phases: Initialization, Processing associated data, Encryption, and Finalization. These phases are executed in the order specified above.

Initialization. In the initialization, the key K and the public message number (nonce) IV are loaded into the three states T_3, T_4, T_6 and the states go through 15 rounds.

$$
\begin{aligned}
&T_3[0] = K; T_3[1] = K; T_3[2] = IV;\\
&T_4[0] = K; T_4[1] = K; T_4[2] = IV; T_4[3] = Z_0;\\
&T_6[0] = K; T_6[1] = K; T_6[2] = IV; T_6[3] = Z_1; T_6[4] = 0; T_6[5] = 0;\\
&\mathsf{for}\ i = 1\ \mathsf{to}\ 15\\
&\quad \mathit{Update}(T_3, T_4, T_6, Z_0, Z_1, Z_0);\\
&\mathsf{end\ for}
\end{aligned}
$$

Processing associated data. Assume the padded associated data has d blocks: $AD = AD_1, \dots, AD_d$. Recall that each block is composed of two words, i.e. $AD_i = AD_i^0 \| AD_i^1$. The Processing associated data is defined as:

$$
\begin{aligned}
&\mathsf{for}\ i = 1\ \mathsf{to}\ d\\
&\quad \mathit{Update}(T_3, T_4, T_6, AD_i^0, AD_i^1, AD_i^0 \oplus AD_i^1);\\
&\mathsf{end\ for}
\end{aligned}
$$

Encryption. Assume the padded message has m blocks: $M = M_1, \dots, M_l$. Recall that each block is composed of two words, i.e. $M_i = M_i^0 \| M_i^1$. In the encryption, a block M_i is processed in one round, and a block of ciphertext $C_i = C_i^0 \| C_i^1$ is output. The encryption is defined as:

$$
\begin{aligned}
&\mathsf{for}\ i = 1\ \mathsf{to}\ m\\
&\quad \mathit{Update}(T_3, T_4, T_6, M_i^0, M_i^1, M_i^0 \oplus M_i^1);\\
&\quad C_i^0 = T_3[0] \oplus T_3[2] \oplus T_4[1] \oplus (T_6[3] \& T_4[3])\\
&\quad C_i^1 = T_6[0] \oplus T_4[2] \oplus T_3[1] \oplus (T_6[5] \& T_3[2])\\
&\mathsf{end\ for}
\end{aligned}
$$

A note about the last ciphertext block. If the last message block of the original, unpadded message is incomplete (the size is less than 32 bytes) and has b bytes ($b < 32$), then the last ciphertext block is truncated to the first b bytes as well – the first b bytes of C_m are kept, while the remaining $32 - b$ are discarded.

The ciphertext C is defined as a concatenation of all ciphertext blocks (with the exception that the last block might be truncated) that have been output during the encryption, i.e. $C = C_1 \| C_2 \| \dots \| C_m$.

Finalization/Tag production. After all message blocks have been processed, the words holding the lengths of the associated data and message are processed, then the states go through 20 more rounds, and the tag Tag is produced as an XOR of all words of all states. This final phase is defined as:

$$
\begin{aligned}
&\mathsf{Update}(T_3, T_4, T_6, AD_{length}, M_{length}, AD_{length} \oplus M_{length});\\
&\text{for } i = 1 \text{ to } 20\\
&\quad \mathsf{Update}(T_3, T_4, T_6, Z_1, Z_0, Z_1);\\
&\text{end for}\\
&Tag = T_3[0] \oplus T_3[1] \oplus T_3[2] \oplus T_4[0] \oplus T_4[1] \oplus T_4[2] \oplus T_4[3] \oplus\\
&\qquad T_6[0] \oplus T_6[1] \oplus T_6[2] \oplus T_6[3] \oplus T_6[4] \oplus T_6[5]
\end{aligned}
$$

1.5.1 Decryption and Verification

In the decryption-verification process, the order of the phases is the same[3]: Initialization, Processing associated data, Decryption, and Finalization. Initialization, Processing associated data and Finalization are the same as during the encryption. Let us define the Decryption. Assume the ciphertext C has m blocks, i.e. $C = C_1 \| C_2 \| \ldots \| C_m$. Then the Decryption phase is defined as:

$$
\begin{aligned}
&\text{for } i = 1 \text{ to } m\\
&\quad \mathsf{Update}(T_3, T_4, T_6, 0, 0, 0);\\
&\quad M_i^0 = C_i^0 \oplus T_3[0] \oplus T_3[2] \oplus T_4[1] \oplus (T_6[3] \& T_4[3])\\
&\quad M_i^1 = C_i^1 \oplus T_6[0] \oplus T_4[2] \oplus T_3[1] \oplus (T_6[5] \& T_3[2]) \oplus M_i^0\\
&\quad T_3[0] = T_3[0] \oplus M_i^0\\
&\quad T_4[0] = T_4[0] \oplus M_i^1\\
&\quad T_6[0] = T_6[0] \oplus M_i^0 \oplus M_i^1\\
&\text{end for}
\end{aligned}
$$

The Decryption might seem less efficient than the Encryption but that is only because in the code above the Update is used. Refer to Appendix A for an efficient implementation of the round of Decryption which uses precisely the same operations as the round of Encryption, and thus has the same efficiency.

If in the Decryption, the last ciphertext block is incomplete, then same as for the Encryption, the block is padded with zero bytes, and the output message block is truncated.

If the produced tag Tag is invalid, the ciphertext and the wrong Tag are not returned.

[3] With the exception that Encryption is replaced by Decryption.

Chapter 2

Security Goals

Table 2.1: Security goals of $\mathtt{Tiaoxin}-346$. There is no secret message number.

goal	bits of security
confidentiality for the plaintext	128
integrity for the plaintext	128
integrity for the associated data	128
integrity for the public message number	128

There is no secret message number. The public message number is a nonce. The goals in table above are under the assumptions that:

1. Each pair of (key, public message number) is used only once[1], nonce-respecting
2. If the verification fails, the ciphertext and the wrong tag are not output[2],

In addition to the above goals, we claim

- Full security against related-key attacks. That is, the adversary is allowed to query chosen messages (plaintexts) under two different related keys and two related (and chosen) nonces, where the relation is given as an XOR difference.
- Full security against distinguishing attacks.
- Full security for messages of any length up to $2^{128}-1$ bits.

[1]Two different pairs of of associated data and plaintext cannot be encrypted under the same pair of key and public message number.

[2]It the verification fails, the only data returned to the user is the value false.

Chapter 3

Security Analysis

3.1 Differential and Linear Trails

High probability differentials for some of the phases of $\mathsf{Tiaoxin}-346$ can lead to forgery and state/key recovery. Thus it is crucial to provide analysis against differential attacks. We do so be investigating the high probability differential trails. More precisely, we are able to find the best differential trails for each of the phases of $\mathsf{Tiaoxin}-346$ by using Matsui's approach [12][1].

The Update operation (more precisely the operation $R(T_s, M)$) has been chosen to assure that the search is feasible and the probability of trails is low. Let us focus on T_3 (the analysis is similar for T_4 and T_6). If in $T_3[0]$ there is a difference, then it must go through 2 AES round calls before it comes again to $T_3[0]$. That is, if at round i there is a difference δ in $T_3[0]$, then at round $i+1$ there is a difference $AES(\delta)$ in $T_3[1]$, at round $i+2$ there is difference $AES(\delta)$ in $T_3[2]$, and finally at round $i+3$ a difference $AES(AES(\delta))$ is added to $T_3[0]$. As there are no other words added between the two AES calls (except the constant Z_0), it means that any difference in $T_3[0]$ will activate 5 S-boxes before coming back in three rounds to $T_3[0]$. This holds as any 2-round AES has 5 active S-boxes (the branch number is 5). Moreover, if at round $i+3$ the difference $AES(AES(\delta))$ is added to $T_3[0]$, and $T_3[0]$ does not contain a difference, then it means in the next three rounds $AES(AES(\delta))$ will go through 2 more AES calls with no other differences added. As a result, the initial difference δ has gone through 4 AES calls, and thus it activated 25 S-boxes. These two observations provide the basis for the search of trails. The probability of the trails found by the search is upper bounded - we assume 2 AES rounds always have 5 active S-boxes (3 always 9, 4 always 25, etc.), but indeed they can have more.

The method to search for trails is the same for all phases, however, the attack frameworks are different. We explain further the cases we have investigated and the results we have obtained.

[1] This approach has been applied several times by the author of $\mathsf{Tiaoxin}-346$ to search for the best related-key trails in various cipher, see [1, 13, 2].

- Initialization: We allow the attacker to inject differences in both the key and the nonce (or only in one of them). The best differential trail on all 15 rounds of the Initialization has 171 active S-boxes (or a probability of at most 2^{-1026}). In fact, after 6 rounds the best trail already has a probability lower than 2^{-256}. This leaves a large security margin against related-key/related-nonce differential attacks. Having such a margin in the Initialization is critical as differential attacks based on trails in some cases can be extended for a few additional rounds.

- Processing associated data and Encryption/Decryption: There are various high probability trails for several rounds of encryptions[2]. However, achieving the initial differences (the starting difference of the trail) in the states is hard, as practically this is equivalent either to finding trails for the Initialization or introducing differences through the messages in the previous rounds. The former has been shown above to be hard, whereas the later cannot be achieved due to the fact that $\mathsf{Tiaoxin}-346$ is nonce-based, and thus the differences in the states are unpredictable (and not zero!).

  One can, however, start with states that do not contain differences – this framework is possible in the Decryption. More precisely, given a associated data, ciphertext and a tag, one can try to build another associated data and ciphertext that results in the same tag. This type of forgery attack has been introduced and exploited in the recent analysis of ALE [5], see [11, 17] for details. Such forgery reduces to the problem of finding a differential trail for the Encryption or the Decryption phases, *that starts and ends with a zero difference in the states*. Our search reveals that the best such trail (see Fig. B.1 of the Appendix) for the Encryption has 30 active S-boxes (six times differences go through 2 AES rounds), or a probability of at most $2^{-6 \cdot 30} = 2^{-180}$. Even if the adversary is able to somehow switch from trails to differentials, the large security margin assures that the trail cannot be exploited as no more than 2^{128} online failed tags can be checked (see the security claims). The best trails for the Decryption are equivalent to the best for Encryption as each of them fixes the difference in the message and in the ciphertext.

- Finalization: Trails for the Finalization do not have to end in a zero difference, however they must start with some non-zero states. The only exception (start with a non-zero difference) is if a difference has been introduced by the first round of the Initialization, i.e. $\mathsf{Update}(T_3,T_4,T_6, AD_{length}, M_{length}, AD_{length} \oplus M_{length})$[3]. The analysis in this case is the same as for the case presented below. In the Finalization, same as in

[2] For instance, there is trail with probability 1 for one round of encryption (any difference in $T_3[1]$ and no differences in the rest of the words of all three states).

[3] This can happen if the last message blocks (or the last associated data blocks) are the same, but have a different length. Due to the padding, the same blocks will be processed during the Encryption (or Processing associated data phase), but M_{length} (or AD_{length}) will be different.

the Initialization, the words input to $\mathtt{Update}$ are constant. If analyzed separately, the best 20-round trail for T_3 has 90 active S-boxes (probability at most 2^{-540}), for T_4 has 80 active (probability at most 2^{-480}), and for T_6 has 62 active S-boxes (at most 2^{-372}). We stress that these are the probabilities only for the Finalization, but if we take into account the probability of getting the initial difference of any of the trails, then the accumulative probability would be much lower. We do not investigate further the exact probability as the current is already very low (below 2^{-372}).

Above, we investigated trails rather than differentials and we provided upper bounds on the probability by counting the number of active S-boxes and assuming each holds with 2^{-6}. The positions of the taps in the output function (i.e. which words are used to produce the ciphertext word) assures that trails are a good estimate of the probability in the Encryption/Decryption phases. This might not be the case for the remaining phases, and here one can work with differentials. However, even in this case the 2-round AES assures low probability (see [14]) and thus $\mathtt{Tiaoxin} - 346$ stays secure.

The security analysis of $\mathtt{Tiaoxin} - 346$ against linear attacks is similar to the above differential analysis – in general high probability linear trails exist in the Encryption/Decryption, however, as the attacker does not have the precise values of the state bytes (only through the ciphertexts bytes, but then they are masked), building exploitable linear trails is hard.

3.2 Rotational Attacks, Internal Differentials and Fixed Points

Rotational attacks [10] can be a threat to AES-based designs as the keyless AES round permutations are susceptible to this type of analysis. In a similar fashion, internal differentials [15], as described in [6] and used in [9], could pose a threat to the security of the design. In $\mathtt{Tiaoxin} - 346$ these two types of attacks are prevented with the use of a non-symmetric constant Z_0 which is added as a subkey to 3 out of the 6 AES calls per $\mathtt{Update}$. If in $T_3[0]$ the internal difference is zero, then after the application of the AES round (and the addition of the subkey Z_0), the internal difference of all 8 bytes will be non-zero. Therefore the next application of the AES round to this word will have very low differential probability (at most 2^{-48}). As a result, after several rounds, the probability of any internal differential trail in T_3 (or even worse, in all three states), will be very low.

Each of the three states has fixed points. For T_3 such point has the form $T_3^F = (X, AES(X) + Z_0, AES(X) + Z_0)$ and a message input $M = AES(AES(X) + Z_0)$, that is $R(T_3^F, M) = T_3^F$. There are $2^{3\cdot128} = 2^{384}$ fixed points for all three states (but only 2^{256} during the encryption due to the requirement $M_2 = M_0 \oplus M_1$). Hence the probability of hitting randomly a fixed point is $2^{384-13\cdot128} = 2^{-1280}$. The initial key and nonce setup in the the Initialization does not permit to start with a fixed point.

3.3 Attacks Based on Similarity of Phases

Note that the three word inputs M_0, M_1, M_2 to Update make each of the phases of $\mathsf{Tiaoxin}-346$ (with the exception of the Processing associated data and Encryption) distinct. In the Initialization ($M_0 = Z_0, M_1 = Z_1, M_2 = Z_0$) and the Finalization ($M_0 = Z_1, M_1 = Z_0, M_2 = Z_1$), M_0, M_1, M_2 do not comply with the condition $M_0 \oplus M_1 = M_2$, whereas this is the case for the middle two phases ($M_0 = M_i^0, M_1 = M_i^1, M_2 = M_i^0 \oplus M_i^1$). This makes the three phases completely different and stops attacks that potentially can exploit similarity of phases, such as slide attacks [3, 4].

3.4 Forgery

To forge the tag, the adversary either has to provide a new tag of previously unencrypted message or to try to find another message that produces some already valid tag. The former reduces to the problem of finding a good differential trail for the Finalization (if the adversary wants to reuse some previous valid tag) or to predict the output of the Finalization with a secret input (the input is secret as the state is unknown). From the above differential analysis we see that the first task cannot be achieved, as the probability of such trails is very low. The second task would require some type of state recovery – a problem analyzed below.

Far more popular approach to forge is by producing collisions in the state and then reusing the already existing valid tag. The collisions can be achieved by chance or on purpose (with differential trails). The big state of $\mathsf{Tiaoxin}-346$ prevents the first type of collisions. The second type are much more dangerous, at least in the Decryption (in the Encryption, this is equivalent to differential trails for the Initialization, which as we have seen above have very low probability). The difference in the Decryption is introduced through associated data and/or the ciphertexts and is canceled in the next several rounds (see [11, 17]). This allows to forge regardless of the Initialization and the Finalization. As long as the adversary can find a high probability trail for the Processing associated data that starts and ends with a zero difference (with non-zero differences in between), she can forge the tag by applying the difference in the associated data. Similarly, if the adversary can find such trails for the Decryption, then she can forge by introducing differences in the ciphertexts. From the differential analysis above we can see that all such trails have low probability, hence this type of forgery cannot be achieved as well.

3.5 State and Key Recovery

There is no bulletproof approach to build an authenticated encryption scheme that guarantees that state recovery is infeasible. Thus for $\mathsf{Tiaoxin}-346$ we can only conjecture that the state recovery requires more effort than a simple brute force of the key. To justify our claim and to deepen the analysis, we add to the following insights:

- Each of the three states of $\mathsf{Tiaoxin} - 346$ holds an information about the key K. That is, if one succeeds in recovering one of the states, then by inverting the design up to the beginning of the initialization, one would be able to recover the key. Therefore for $\mathsf{Tiaoxin} - 346$ a state recovery (one of T_3, T_4, T_6) is equivalent to a key recovery. The values for the words of a particular state, however, are never output straightforwardly, but as XOR (or AND) of words of all three states (refer to the output function used for producing the ciphertexts).

- The three states require different number of rounds to achieve a full diffusion: T_3 needs 5 rounds, T_4 needs 7 rounds, and T_6 needs 11 rounds.

- A single bit of the ciphertext depends on 5 different bits from the states T_3, T_4, T_6. As AND is used in the output function, we can replace $x\&y$ with the constant 0, with probability 3/4 per bit. Hence with probability $(3/4)^{128} \approx 2^{-53.1}$ one can assume that $C_i^0 = T_3[0] \oplus T_3[2] \oplus T_4[1]$ (or $C_i^1 = T_6[0] \oplus T_4[2] \oplus T_3[1]$). This strategy allows to cancel as much as 2 full-state ANDs (if 3 then the probability becomes $2^{-3 \cdot 53.1} = 2^{-159.3} < 2^{-128}$) in arbitrary chosen ciphertext words. However, the non-linear system of equations, obtained in the case of 2 AND cancellations is underdefined as C_i^0, C_{i+1}^0 depend on 5 words: the values of $T_3[0], T_3[1], T_3[2], T_4[0], T_4[1]$ at round i. Therefore, bruteforcing the free variables of the system will result in a complexity worse than key enumeration. If we take system composed of 1 equation we get 3 variables $T_3[0], T_3[2], T_4[1]$, and again the situation is the same: too many bits to guess. Reducing the number of variables in the system as well cannot be achieved if one takes C_i^1 (the second word in a block of ciphertext) – if fact, in this case the number of unknown words only increases.

  One can replace AND with XOR (i.e. $x\&y = x \oplus y \oplus 1$), with the same probability of 3/4 per bit. A motivation for such strategy is the fact that linear combinations of some ciphertexts words could cancel more state words (and lead to system with less variables) than when AND is replaced with 0. This however does not happen, and the system of any 2 (or less) equations has even more free variables.

  Another promising attack strategy, based on the same idea of cancellation of AND (or replacing it with XOR) and then solving a system of equations is as follows. Assume that instead of trying to solve a system where the unknown variables are words, one deduces equations based on bytes. The basis of this idea lies on the fairly weak diffusion of the AES round calls used in Update and of that between the state words. Hence it might be that some ciphertext bytes depend on a smaller number of bytes of the states, i.e. not all ciphertext bytes (output in a few consecutive rounds) depend on all state bytes. Let us try to minimize the number of such bytes and focus only on C_i^0 (with the AND cancellation, C_i^0 depends only on T_3, T_4, while C_i^1 depends on all three). If we choose a single byte of C_i^0 (say at position $(1, 1)$), then it depends on three bytes of $T_3[0], T_3[2], T_4[1]$ at the same position at round i (denote the states as T_3^i, T_4^i). Since we want to minimize the number of free variables in the system, we take the very same byte at the next C_{i+1}^0, but then due to the diffusion, $T_3^{i+1}[0]$ depends on 4 bytes of $T_3^i[2]$, and thus at round i we should have guessed more bytes. Thus we cannot succeed by guessing only a single byte in the states. A similar situation occurs if we focus on one column (rather than byte) of C_i^0 (than the whole word $T_3^i[2]$ should be guessed). Thus we must guess all bytes of C_i^0 at round i and we end up with a scenario that has been discussed above.

3.6 No Claims

The security of $\mathsf{Tiaoxin} - 346$ is analyzed in the framework of authenticated encryption, and we do NOT claim security in any of the more general frameworks/cases such as:

- If the nonce is reused. Obviously in this case high probability trails (that do not need to end in a zero difference) for the Encryption of $\mathsf{Tiaoxin}-346$ can be used to recover state bytes and to compromise the confidentiality.
- In the open key model. When the key is known to the attacker, for instance, colliding tags can be easily produced and various distinguisher are possible.
- If state sizes are different. The state sizes are chosen with a purpose to be 3,4, and 6 words. For instance, if two of the state sizes are equal, then high probability differential trails that end with a zero difference (and that can be used to forge tags), exist for the decryption.
- If there are only two states. Same as above, good trails exist.
- Reduced rounds in the Initialization/Finalization.

To summarize, we claim no additional security to the one specified in Section 2.

Chapter 4

Features

$\mathtt{Tiaoxin}-346$ has the following features:

- It is a nonce-based, software oriented design based on a stream cipher.
- It is the first authenticated encryption (or even MAC) scheme[1] to use only 3 AES rounds per 16-byte message. More precisely, it uses 6 AES round calls per 32-byte message. All 6 calls are fully parallelizable.
- It achieves 0.28 cpb on Intel Haswell. Depending on the software platform (Intel Haswell or Intel Sandy Bridge), it is 3.5 to 6.5 times faster than the benchmark AES-GCM, and twice faster than OCB3 (or AES-128 in counter mode). The speed can be improved further if latency of AES-NI $\mathtt{aesenc}$ is reduced and the throughput stays the same[2].
- It is analyzed against various types of attacks. Most of the design decisions were made in order to make the cipher secure. The security claims are maximum expected in the framework of nonce-respecting adversaries. It provides full security against related-key attacks.
- Accepts very long messages of sizes up to $2^{128}-1$ bits. No loss of security on long messages.
- State sizes found to be optimal among the all state sizes following the design strategy and being secure.

[1] To the best of our knowledge.

[2] This occurred with the introduction of Haswell, where compared to Sandy/Ivy Bridge, the throughput stayed the same but the latency got reduced by one cycle.The current latency on Haswell is 7 cycles; $\mathtt{Tiaoxin}-346$ would run even faster if the latency was 6 cycles.

Chapter 5

Design Rationale

$\mathsf{Tiaoxin}-346$ is designed to achieve maximal software efficiency while maintaining high level of security. The main transformation used is the AES round function, while parts of the design are inspired by AEGIS [16].

All the design elements used in $\mathsf{Tiaoxin}-346$ serve a particular purpose and were chosen either to increase the efficiency or to increase the level of security:

- Parallel AES round calls instead of serial. The AES-NI instruction for performing one round of AES increases significantly the efficiency of AES-based designs, but this instruction has rather large latency. That is, the result of this operation cannot be used for some number of cycles (8 for Intel Sandy Bridge, 7 for Haswell), and thus designs based on serial execution of AES calls are not that fast unless multiple messages are processed at the same time. The best example is the comparison of the speed of AES in cipher block chaining (serial) and counter (parallel) modes: the second runs around 7-8 times faster. In $\mathsf{Tiaoxin}-346$ we parallelize all AES calls within one round of Update, and hence the speed of the design is very high – on average, to execution the 6 AES calls in Update requires only 7-8 cycles on the new Intel processors (around $0.22-0.25$ cycles per byte go for the AES calls in $\mathsf{Tiaoxin}-346$). To achieve this parallelization, we had to increase the state size.

- Three states. Each of the first two states T_3, T_4 processes independently message words. The transformations used in these two states are not sufficiently strong to assure full security as each of them uses only 2 AES round calls per 16-byte message. However, the third state processes all of the words from the first two states, and significantly increases the security of the whole design. This is the main trick that allows to reduce the number of AES calls per 16-byte message to only 3.

- State sizes. No two state sizes (among the three) should be the same as otherwise high probability differential trails exist. For instance, assume that the first two states have the same size. Then exist same differential trails for both of them, which result in no difference for the third state (because the inputs to the third is $M_i^0 \oplus M_i^1$, thus the differences cancel). This would immediately lead to forgery attacks. With a computer search we found that the smallest state cannot be less than 3 three words while the design is still secure. The states of sizes 3,4,5 were not chosen as high probability differential trail exist (again found with computer search) for such states – the trail is presented in Fig. C.1 of the Appendix). Hence the chosen state sizes are optimal (smallest and secure) among all we have tried.

- No diffusion between states. This significantly simplifies the analysis and we are able to deterministically find the best differential trails. We have also investigated possible designs that have some inter state diffusion. However, our partial brute force search did not return any secure candidates.

- Stronger diffusion in the states. We have also investigated state operations that have stronger diffusion than the currently used. Such designs are much harder to analyze and also have reduced efficiency. The main reason for discarding most of such designs, however, is a complete loss of a reasonable security proof. In each of our current states, between two AES calls there is only an addition of the constant Z_0 (for instance, in T_3, the word $T_3[0]$ first goes through AES round with Z_0 as subkey and then again goes through AES round, with no other words added in between the two AES calls). This assures that once there is a difference in $T_3[0]$ (or $T_4[0], T_6[0]$), the number of active S-boxes activated by this difference, by the time it gets again to the word at position 0, must be at least 5 (because the branch number of AES is 5). Hence to find the best trails we only have to count how many times the words at position 0,in all three states, have differences. If the diffusion in the states was stronger, and there was a word added between the two AES calls, then the probability of the differential trails will rise significantly – instead of working with 2-round AES, we will have to deal with 1-round AES which has as low as 1 active S-box for some trails.

- The output function for producing the ciphertext. The taps (the position of the words used for producing the ciphertext words) in the output function were chosen to make the state recovery harder – it assures that no linear transformation of few consecutive ciphertexts words reveal any of the state words. AND (&) in the output function was chosen to stop the potential state recovery. We note that there exist several other combinations of secure tap positions and that AND can be replaced by OR.

- The use of constants Z_0, Z_1. These two constants are the first four 64-bit words used as constants in SHA-512. Their main purpose is to stop rotational and internal differential attacks, and, in general, attacks based on similarity of operations. The constants are used to make the Initialization and Finalization of $\mathsf{Tiaoxin}-346$, with two different key sizes, completely different.

- Constants as inputs to Initialization, Finalization. In these two phases the Update operation has constant word inputs. This assures that all differential trails will have very low probability (as differences in the states cannot be canceled) and they cannot be exploited to attack the design. The word inputs (which are either the constant Z_0 or the constant Z_1) do not comply to the requirement of the third word being XOR of the first two (which is the case for the Encryption phase). This makes the phases different from the Encryption and stops slide attacks.

Chapter 6

Software Performance

$\mathsf{Tiaoxin}-346$ is designed to take a full advantage of the dedicated AES instruction set implemented in the latest processors. In particular, the design extensively uses the AES-NI instruction aesenc which is one keyed encryption round of AES. The Update operation basically is the only transformation used in $\mathsf{Tiaoxin}-346$, and can be implemented with only 6 calls to aesenc and 3-4 XORs. Each of the calls can be executed independently, and since the latency of aesenc is at best 7 cycles (achieved on the latest Intel processor Haswell), there is no stall in executing these calls. In fact, $\mathsf{Tiaoxin}-346$ theoretically could be even more efficient if the latency of aesenc was 6 cycles. The low number of AES calls per message block guarantees that even a simple table look-up implementation would result in an efficient cipher.

The output function used for producing the ciphertexts in the encryption phase requires 6 XORs and 2 ANDs. The Initialization and Finalization bear an overhead which is equivalent to encrypting 1120 bytes of message. This reflects on the speed measurements for short messages, however, even in this case the speed of $\mathsf{Tiaoxin}-346$ is very high (see Tbl 6.1).

To test the actual efficiency of the design, we have implemented the authenticated encryption on C with the support of AES intrinsics and obtained benchmarks on two platforms. The first platform is Intel Core i5-2467M 1.6GHz (Sandy Bridge) with 64-bit Linux Mint 16 Petra and gcc 4.8.1. The second is Intel Core i5-4570 CPU 3.20GHz (Haswell) with 64-bit Linux Mint 16 Petra and gcc 4.8.1. On both, the code is compiled with the switches "-O3 -msse2 -maes -mavx -march=native", and Turbo Boost is off when running. To obtain more precise results, we took the average speed over 1000 experiments. In each of the experiments we took randomly chosen key and nonce. The message in the first experiment was chosen randomly, and in the following experiment the ciphertext was used as the next message (with stores to and loads from memory, in between). The results of the testing along with the benchmarks of AES-128 in counter mode, AES-128 OCB3, and AES-128-GCM are given in Tbl. 6.1. We note that the entries in the table for these three schemes were taken from [5, 8]. In the cases when the papers reported different benchmarks for the same design, we took the more efficient estimate. Also, we could not find the speed for all message lengths (long messages of around 64KB, etc.) – in this case we assumed that the speed is the same as for shorter messages – thus the results are presented with question marks.

On long messages, $\mathsf{Tiaoxin}{-}346$ runs at round 0.38 cycles per byte on Sandy Bridge. It is two times faster than AES, and 6-7 times faster than AES-128-GCM, which is the supposed CAESAR benchmark authenticated encryption scheme. On Haswell $\mathsf{Tiaoxin} - 346$ runs at 0.28 and again is around two times faster than AES-128 in counter mode. The speed advantage compared to AES-128-GCM is smaller, however, $\mathsf{Tiaoxin} - 346$ is still around 3.5 times faster.

We have also benchmarked the authentication part alone in $\mathsf{Tiaoxin} - 346$– in this case there is no overhead of the output function and the design runs much faster. On Sandy Bridge to authenticate 64KB long messages, $\mathsf{Tiaoxin} - 346$ needs around 0.27 cycles per byte, while on Haswell around 0.25 cycles per byte.

Table 6.1: The speed of (authenticated) encryption schemes compared to the speed of $\mathsf{Tiaoxin} - 346$ (encryption + authentication). All measurements are given in cycles per byte. The numbers in the second row are the length of the message inputs in bytes. All the measurements except for $\mathsf{Tiaoxin} - 346$ are taken from [5, 8]. In [8] most of the measurements do not specify the message bytes – "?" is used to signal this in the table.

	Intel Sandy Bridge								Intel Haswell	
	128	256	512	1024	2048	4096	8192	2^{16}	8192	2^{16}
AES-128-CTR	1.61	1.22	0.99	0.87	0.80	0.77	0.76	0.73?	0.63?	0.63?
AES-128-OCB3	2.69	1.79	1.34	1.12	1.00	0.88	0.86	0.86?	0.69?	0.69?
AES-128-GCM	4.95	3.88	3.33	3.05	2.93	2.90	2.55	2.53?	1.03	1.03?
$\mathsf{Tiaoxin} - 346$	2.49	1.45	0.91	0.65	0.50	0.44	0.40	0.38	0.31	0.28

Chapter 7

Intellectual property

$\mathsf{Tiaoxin}-346$ is not patented and is free for use in any application.

If any of this information changes, the submitter/submitters will promptly (and within at most one month) announce these changes on the crypto-competitions mailing list.

Chapter 8

Consent

The submitter/submitters hereby consent to all decisions of the CAESAR selection committee regarding the selection or non-selection of this submission as a second-round candidate, a third-round candidate, a finalist, a member of the final portfolio, or any other designation provided by the committee. The submitter/submitters understand that the committee will not comment on the algorithms, except that for each selected algorithm the committee will simply cite the previously published analyses that led to the selection of the algorithm. The submitter/submitters understand that the selection of some algorithms is not a negative comment regarding other algorithms, and that an excellent algorithm might fail to be selected simply because not enough analysis was available at the time of the committee decision. The submitter/submitters acknowledge that the committee decisions reflect the collective expert judgments of the committee members and are not subject to appeal. The submitter/submitters understand that if they disagree with published analyses then they are expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions. The submitter/submitters understand that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

Acknowledgements

I would like to thank Jian Guo, Jérémy Jean, Tetsu Iwata, Thomas Peyrin, Lei Wang and Hongjun Wu for helpful discussions and suggestions. This work is sponsored by the Singapore National Research Foundation Fellowship 2012 NRF-NRFF2012-06.

Bibliography

[1] A. Biryukov and I. Nikolić. Automatic search for related-key differential characteristics in byte-oriented block ciphers: Application to AES, Camellia, Khazad and others. In H. Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 322–344. Springer, 2010.

[2] A. Biryukov and I. Nikolić. Search for related-key differential characteristics in DES-like ciphers. In A. Joux, editor, *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2011.

[3] A. Biryukov and D. Wagner. Slide attacks. In L. R. Knudsen, editor, *FSE*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 1999.

[4] A. Biryukov and D. Wagner. Advanced slide attacks. In B. Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 589–606. Springer, 2000.

[5] A. Bogdanov, F. Mendel, F. Regazzoni, V. Rijmen, and E. Tischhauser. ALE: AES-based lightweight authenticated encryption. *20th International Workshop on Fast Software Encryption–FSE*, 2013. To appear.

[6] I. Dinur, O. Dunkelman, and A. Shamir. Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials. FSE, 2013. To appear.

[7] S. Gueron. Intel advanced encryption standard (AES) instructions set. *Intel White Paper, Rev*, 3, 2010.

[8] S. Gueron. AES-GCM software performance on the current high end CPUs as a performance baseline for CAESAR competition. *DIAC 2013: Directions in Authenticated Ciphers*, 2013.

[9] J. Guo, I. Nikolić, T. Peyrin, and L. Wang. Cryptanalysis of Zorro. Cryptology ePrint Archive, Report 2013/713, 2013. `http://eprint.iacr.org/`.

[10] D. Khovratovich and I. Nikolić. Rotational cryptanalysis of ARX. In S. Hong and T. Iwata, editors, *FSE*, volume 6147 of *Lecture Notes in Computer Science*, pages 333–346. Springer, 2010.

[11] D. Khovratovich and C. Rechberger. The LOCAL attack: Cryptanalysis of the authenticated encryption scheme ALE. SAC, 2013. To appear.

[12] M. Matsui. On correlation between the order of S-boxes and the strength of DES. In A. D. Santis, editor, *EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 366–375. Springer, 1994.

[13] I. Nikolić. Tweaking AES. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 198–210. Springer, 2010.

[14] S. Park, S. H. Sung, S. Lee, and J. Lim. Improving the upper bound on the maximum differential and the maximum linear hull probability for SPN structures and AES. In T. Johansson, editor, *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 247–260. Springer, 2003.

[15] T. Peyrin. Improved differential attacks for ECHO and Grøstl. In T. Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 370–392. Springer, 2010.

[16] H. Wu and B. Preneel. AEGIS: A fast authenticated encryption algorithm. Cryptology ePrint Archive, Report 2013/695, 2013. `http://eprint.iacr.org/`.

[17] S. Wu, H. Wu, T. Huang, M. Wang, and W. Wu. Leaked-state-forgery attack against the authenticated encryption algorithm ALE. In K. Sako and P. Sarkar, editors, *ASIACRYPT*, volume 8269 of *Lecture Notes in Computer Science*, pages 377–404. Springer, 2013.

Appendix A

Optimal Software Implementation of a Round of Decryption

One complete round of Encryption, uses 6 keyed AES calls aesenc and 4 XORs in Update, with an additional 6 XORs and 2 ANDs in the output function – in total 6 AES call, 10 XORs and 2 ANDs. A round of Decryption can be implemented with the very same number and type of operations:

$T_3^{new}[1] = AES(T_3[0], Z_0); T_3^{new}[2] = T_3[1]$
$T_4^{new}[1] = AES(T_4[0], Z_0); T_4^{new}[2] = T_4[1]; T_4^{new}[3] = T_4[2]$
$T_6^{new}[1] = AES(T_6[0], Z_0); T_6^{new}[2] = T_6[1]; T_6^{new}[3] = T_6[2]; T_6^{new}[4] = T_6[3]; T_6^{new}[5] = T_6[4]$
$T_3^{new}[0] = C_i^0 \oplus T_3^{new}[2] \oplus T_4^{new}[1] \oplus (T_6^{new}[3] \& T_4^{new}[3])$
$M_i^0 = AES(T_3[2], T_3[0]) \oplus T_3^{new}[0]$
$T_6[0]^{new} = C_i^1 \oplus T_4^{new}[2] \oplus T_3^{new}[1] \oplus (T_6^{new}[5] \& T_3^{new}[2])$
$M_i^1 = AES(T_6[5], T_6[0]) \oplus T_6^{new}[0] \oplus M_i^0$
$T_4[0]^{new} = AES(T_4[3], T_4[0]) \oplus M_i^1$
$T_3 = T_3^{new}; T_4 = T_4^{new}; T_5 = T_5^{new}$

Appendix B

The Best Differential Trail for $\mathsf{Tiaoxin}-346$

The best differential trail has 30 active S-boxes (a difference goes 6 times through 2 AES rounds) and holds with a probability of at most 2^{-180}. The trail is given in Fig. B.1.

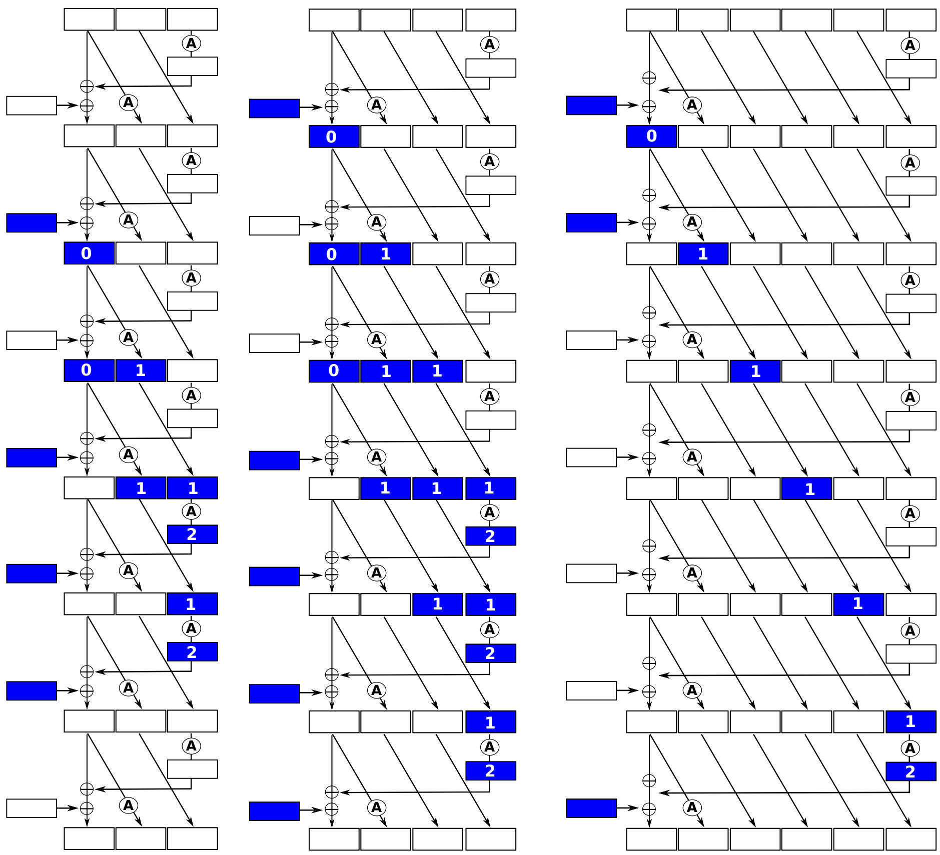

Figure B.1: A best differential trail for Tiaoxin − 346. The blue rectangles, denote words with differences. The number in the state words (in the blue rectangles), denotes how many times a difference went through AES rounds without some other word difference being added to it.

Appendix C

On the Authenticated Encryption with States with Sizes 3,4,5

As mentioned earlier, we have not chosen states of sizes 3,4,5, as in this case there exist a high probability differential trail that can be used to produce forgery. The trail is given in Fig. C.1, has 20 active S-boxes, and holds with a probability of 2^{-120}.

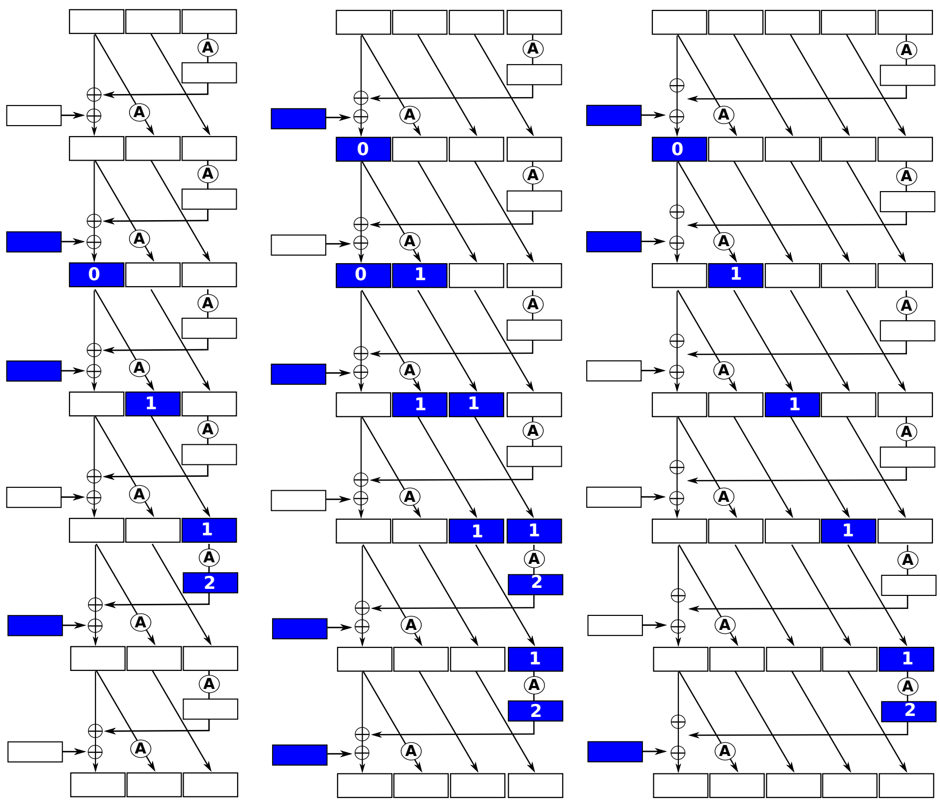

Figure C.1: A differential trail when the states have sizes 3,4,5. The blue rectangles, denote words with differences. The number in the state words (in the blue rectangles), denotes how many times a difference went through AES rounds without some other word difference being added to it.