ACORN:
A Lightweight Authenticated Cipher
(v2)

Designer and Submitter: Hongjun Wu

Division of Mathematical Sciences
Nanyang Technological University
wuhongjun@gmail.com

2015.08.29

Contents

Chapter 1

Specification

The specifications of ACORN-128 are given in this chapter.

1.1 Recommended parameter sets

- Primary Recommendation: ACORN-128
  128-bit key, 128-bit nonce, 128-bit tag

1.2 Operations, Variables and Functions

The operations, variables and functions used in ACORN are defined below.

1.2.1 Operations

The following operations are used in ACORN:

$\oplus$	:	bit-wise exclusive OR
&	:	bit-wise AND
$\sim$	:	bit-wise NOT
$\|\|$	:	concatenation
$\lceil x \rceil$	:	ceiling operation, $\lceil x \rceil$ is the smallest integer not less than x

1.2.2 Variables and constants

The following variables and constants are used in ACORN:

AD	:	associated data (this data will not be encrypted or decrypted).
ad_i	:	one bit of associated data block.
$adlen$	:	bit length of the associated data with $0 \leq adlen < 2^{64}$.
C	:	ciphertext.
c_i	:	one ciphertext bit.

ca_i	:	a control bit at the ith step. It is used to separate the processing of associated data, the processing of plaintext, and the generation of authentication tag.
cb_i	:	another control bit at the ith step. It is used to allow a keystream bit to affect a feedback bit during initialization, processing of associated data, and the tag generation.
IV_{128}	:	128-bit initialization vector of ACORN-128.
$IV_{128,i}$	:	the ith bit of IV_{128}.
K_{128}	:	128-bit key of ACORN-128.
$K_{128,i}$	:	the ith bit of K_{128}.
ks_i	:	The keystream bit generated at the ith step.
$pclen$	:	bit length of the plaintext/ciphertext with $0 \leq pclen < 2^{64}$.
m_i	:	one data bit.
P	:	plaintext.
p_i	:	one plaintext bit.
S_i	:	state at the beginning of the ith step.
$S_{i,j}$	:	jth bit of state S_i . For ACORN-128, $0 \leq j \leq 292$.
T	:	authentication tag.
t	:	bit length of the authentication tag with $64 \leq t \leq 128$.

1.2.3 Functions

Two Boolean functions are used in ACORN: maj and ch.

$$\begin{aligned} maj(x, y, z) &= (x\&y) \oplus (x\&z) \oplus (y\&z) \ ; \\ ch(x, y, z) &= (x\&y) \oplus ((\sim x)\&z) \ ; \end{aligned}$$

1.3 ACORN-128

ACORN-128 uses a 128-bit key and a 128-bit initialization vector. The associated data length and the plaintext length are less than 2^{64} bits. The authentication tag length is less than or equal to 128 bits. We strongly recommend the use of a 128-bit tag.

1.3.1 The state of ACORN-128

The state size of ACORN-128 is 293 bits. There are six LFSRs being concatenated in ACORN-128. The state is shown in Fig.1.1.

1.3.2 The functions of ACORN-128

There are three functions in ACORN-128: the function to generate keystream bit from the state, the function to compute the overall feedback bit, and the function to update the state.

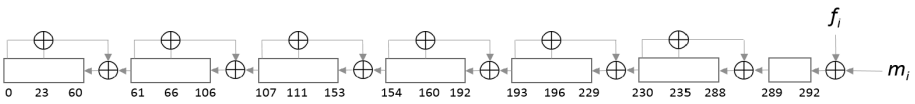

Figure 1.1: The concatenation of 6 LFSRs in ACORN-128. f_i indicates the overall feedback bit for the ith step; m_i indicates the message bit for the ith step.

Generate the Keystream Bit. At each step, the keystream bit is computed using the function $ks_i = KSG128(S_i)$:

$$ks_i = S_{i,12} \oplus S_{i,154} \oplus maj(S_{i,235}, S_{i,61}, S_{i,193}) \ ;$$

Compute the Feedback Bit. At each step, the feedback bit is computed using the function $f_i = FBK128(S_i, ca_i, cb_i)$:

$$\begin{aligned} ks_i &= KSG128(S_i) \ ; \\ f_i &= S_{i,0} \oplus (\sim S_{i,107}) \oplus maj(S_{i,244}, S_{i,23}, S_{i,160}) \oplus ch(S_{i,230}, S_{i,111}, S_{i,66}) \oplus \\ &\quad (ca_i \& S_{i,196}) \oplus (cb_i \& ks_i) \ ; \end{aligned}$$

The State Update Function. At each step, the pseudo code for the state update function $S_{i+1} = \text{StateUpdate128}(S_i,\ m_i,\ ca_i,\ cb_i)$ is given as :

$$\begin{aligned} S_{i,289} &= S_{i,289} \oplus S_{i,235} \oplus S_{i,230}; \\ S_{i,230} &= S_{i,230} \oplus S_{i,196} \oplus S_{i,193}; \\ S_{i,193} &= S_{i,193} \oplus S_{i,160} \oplus S_{i,154}; \\ S_{i,154} &= S_{i,154} \oplus S_{i,111} \oplus S_{i,107}; \\ S_{i,107} &= S_{i,107} \oplus S_{i,66} \oplus S_{i,61}; \\ S_{i,61} &= S_{i,61} \oplus S_{i,23} \oplus S_{i,0}; \end{aligned}$$

$f_i = FBK128(S_i, ca_i, cb_i)$;

for $j := 0$ to 291 do $S_{i+1,j} = S_{i,j+1}$;

$S_{i+1,292} = f_i \oplus m_i$;

1.3.3 The initialization of ACORN-128

The initialization of ACORN-128 consists of loading the key and IV into the state, and running the cipher for 1792 steps.

1. Initialize the state S_{-1792} to 0.
2. Let $m_{-1792+i} = K_{128,i}$ for $i = 0$ to 127;
   Let $m_{-1792+128+i} = IV_{128,i}$ for $i = 0$ to 127;

Let $m_{-1792+256} = K_{128,i \mod 128} \oplus 1$ for $i = 0$;
Let $m_{-1792+256+i} = K_{128,i \mod 128}$ for $i = 1$ to 1535;

3. Let $ca_{-1792+i} = 1$ for $i = 0$ to 1791;
   Let $cb_{-1792+i} = 1$ for $i = 0$ to 1791;

4. for $i = -1792$ to -1, $S_{i+1} = \text{StateUpdate128}(S_i, m_i, ca_i, cb_i)$;

Note that in the initialization, the keystream bit is used to update the state since $cb_i = 1$.

1.3.4 Processing the associated data

After the initialization, the associated data AD is used to update the state.

1. Let $m_i = ad_i$ for $i = 0$ to $adlen - 1$;
   Let $m_{adlen} = 1$;
   Let $m_{adlen+i} = 0$ for $i = 1$ to 255;

2. Let $ca_i = 1$ for $i = 0$ to adlen+127;
   Let $ca_i = 0$ for $i =$ adlen+128 to adlen+255;
   Let $cb_i = 1$ for $i = 0$ to adlen+255;

3. for $i = 0$ to $adlen + 255$, $S_{i+1} = \text{StateUpdate128}(S_i, m_i, ca_i, cb_i)$;

Note that even when there is no associated data, we still need to run the cipher for 256 steps. When we process the associated data, the keystream bit is used to update the state since $cb_i = 1$. The cipher specification is changed for 128 steps (since the value of ca_i is set to 0 for 128 steps) so as to separate the associate data from the plaintext/ciphertext.

1.3.5 The encryption

After processing the associated data, at each step of the encryption, one plaintext bit p_i is used to update the state, and p_i is encrypted to c_i.

1. Let $m_{adlen+256+i} = p_i$ for $i = 0$ to $pclen - 1$;
   Let $m_{adlen+256+pclen} = 1$;
   Let $m_{adlen+256+pclen+i} = 0$ for $i = 1$ to 255;

2. Let $ca_i = 1$ for $i = adlen + 256$ to $adlen + pclen + 383$;
   Let $ca_i = 0$ for $i = adlen + pclen + 384$ to $adlen + pclen + 511$;
   Let $cb_i = 0$ for $i = adlen + 256$ to $adlen + pclen + 511$;

3. for $i = adlen + 256$ to $adlen + pclen + 511$,
   $S_{i+1} = \text{StateUpdate128}(S_i, m_i, ca_i, cb_i)$;
   $c_i = p_i \oplus KSG128(S_i)$;
   end for;

Note that even when there is no plaintext, we still need to run the cipher for 256 steps. When we process the plaintext, the keystream bit is not used to update the state since $cb_i = 0$. The cipher specification is changed for 128 steps (since the value of ca_i is set to 0 for 128 steps) so as to separate the processing of plaintext/ciphertext and the finalization.

1.3.6 The finalization

After processing all the plaintext bits, we generate the authentication tag T.

1. Let $m_{adlen+pclen+512+i} = 0$ for $i = 0$ to 767;
2. Let $ca_i = 1$ for $i = adlen + pclen + 512$ to $adlen + pclen + 1279$;
   Let $cb_i = 1$ for $i = adlen + pclen + 512$ to $adlen + pclen + 1279$;
3. for $i = adlen + pclen + 512$ to $adlen + pclen + 1279$,
   $S_{i+1} = \text{StateUpdate128}(S_i, m_i, ca_i, cb_i)$;
   $ks_i = KSG128(S_i)$;
   end for;

The authentication tag T is the last t keystream bits, i.e.,
$T = ks_{adlen+pclen+1279-t+1} \parallel ks_{adlen+pclen+1279-t+2} \parallel \cdots \parallel ks_{adlen+pclen+1279}$.

1.3.7 The decryption and verification

The decryption and verification are very similar to the encryption and tag generation. The finalization in the decryption process is the same as that in the encryption process. We emphasize that if the verification fails, the ciphertext and the newly generated authentication tag should not be given as output; otherwise, the state of ACORN-128 is vulnerable to known-plaintext or chosen-ciphertext attacks (using a fixed IV).

Chapter 2

Security Goals

The security goals of ACORN are given in Table 2.1. In ACORN, each key, IV pair is used to protect only one message. If verification fails, the new tag and the decrypted ciphertext should not be given as output.

Note that the authentication security in Table 2.1 includes the integrity security of plaintext, associated data and nonce.

Table 2.1: Security Goals of ACORN-128 (128-bit tag)

	Encryption	Authentication
ACORN-128	128-bit	128-bit

Chapter 3

Security Analysis

The following requirements should be satisfied in order to use ACORN securely.

1. Each key should be generated in a secure and random way.
2. Each key and IV pair should not be used to protect more than one message; and each key and IV pair should not be used with two different tag sizes.
3. If verification fails, the decrypted plaintext and the wrong authentication tag should not be given as output.

If the above requirements are satisfied, we have the following security claims:

Claim 1. The success rate of a forgery attack is 2^{-t}, where t is the tag size. If the forgery attack is repeated n times, the success rate of a forgery attack is about $n \times 2^{-t}$.

Claim 2. The state and key cannot be recovered faster than exhaustive key search if the forgery attack is not successful. We recommend the use of a 128-bit tag size for ACORN in order to resist repeated forgery attacks.

If an IV is reused in encryption, or if the plaintext is leaked in the failed verification, the state can be recovered easily. In [2], it is shown that if the IV is reused seven times, the security of ACORN is lost. We point out here that in the new version of ACORN, the secret key of ACORN cannot be recovered easily from the state, so now ACORN provides low-level resilience against the nonce reuse attack.

According to our analysis, ACORN is a strong cipher. Since the design approach of ACORN is very new, we encourage the researchers to conduct thorough security analysis of ACORN.

In this chapter, we mainly analyze the security of authentication, since it is very challenging to design and analyze the differential propagation in an authenticated cipher based on a sequential stream cipher.

3.1 The security of the initialization

The initialization can be attacked by analyzing the relation between IV and keystream. In ACORN-128, the IV passes through at least 1792 steps before affecting ciphertext. This large number of steps in the initialization is to prevent various attacks against stream cipher initialization: the linear attack (such as the attack in [14]), differential attacks (such as the attacks in [16] and [15]) and cube attacks [7, 8].

3.2 The security of the encryption process

We emphasize here that ACORN encryption is a stream cipher with a large state which is updated continuously. The attacks against a block cipher cannot be applied directly to ACORN.

Statistical Attacks. If the IV is used only once for each key, it is impossible to apply a differential attack to the encryption process. It is extremely difficult to apply a linear attack (or correlation attack) to recover the secret state since the state of ACORN is updated in a nonlinear way. In general, it would be difficult to apply any statistical attack to recover the secret state due to the nonlinear state update function (the statistical correlation between any two states vanishes quickly as the distance between them increases).

3.3 The security of message authentication

A common approach to attack ACORN authentication is to inject a difference into the state by modifying ciphertext or associated data. Ensuring the security of authentication is the most challenging part in the design and security analysis of ACORN.

A main feature of ACORN-128 is the concatenation of 6 small LFSRs, as shown in Fig. 1.1. The concatenation of six LFSRs ensures that once a difference bit is injected into the state (the first difference bit must be injected into the state through m_i), there are many difference bits in the state before the state difference gets eliminated.

To eliminate the difference in the right most LFSR, the input difference to that LFSR should have the following linear recurrence (in order to reduce the number of difference bits in the state, we consider only the shortest linear recurrence):

$$d_n = d_{n-59} \oplus d_{n-53}$$

Similarly, in order to eliminate the difference in each of the other five LFSRs, the input difference should have the following linear recurrences:

$$d_n = d_{n-37} \oplus d_{n-34}$$
$$d_n = d_{n-39} \oplus d_{n-33}$$

$$d_n = d_{n-47} \oplus d_{n-43}$$
$$d_n = d_{n-46} \oplus d_{n-41}$$
$$d_n = d_{n-61} \oplus d_{n-38}$$

Combining the above six linear occurrences, the input difference to those six registers is given below. There are other input differences, but this difference is likely the shortest one (290 bits).

```
10000000000000000000000000000000011001110101001100000100000101000
00010011111000100011100111110001010110001011101011111011011100101
01000110001010101111101010110110001000010011100000100001000110011001
00010110010100100100010000101011101001001010101110111001111011000
01110111011011101110000011101110011
```

Once the input difference to those six registers is known, we are able to compute the probability that the difference in the state can be eliminated. Note that the difference in the overall feeback bit f_i can always get eliminated with certain probability by modifying the ciphertext or associated data.

The analysis of the difference in associated data and the analysis of the difference in ciphertext are similar (except that the ciphertext leaks the state information which is useful for the attack). The reason is that when the associated data gets processed, the keystream bit is used as part of the feedback bit; when the ciphertext gets processed, the keystream is generated and xored to the ciphertext, then the decrypted plaintext bit is used as part of the feedback bit (i.e., the keystream bit also affects the input difference to the LFSRs).

We note that at each step, there are three nonlinear functions being involved: two maj functions and one ch function. The differential property of function maj is that if there are one or two input difference bits, then the output difference is 1 with probability 0.5; if there are three input difference bits, then the output difference is 1 with probability 1. The differential property of function $ch(x, y, z)$ is that if there are differences in both y and z, then the output difference is one with probability 1; otherwise, if there is any difference in the input, the output difference is 1 with probability 0.5.

According to the above input difference to the LFSRs and the differential properties of the nonlinear functions, the probability to eliminate the difference in the state is 2^{-189} (in a successful attack, proper difference should be injected into associated data or ciphertext so that the input difference to LFSRs is not affected by the nonlinear functions). With this differential probability, ACORN is able to provide 128-bit MAC security.

Chapter 4

Features

- Novel design. ACORN is a sequential authenticated cipher which is efficient in hardware and software, and its authentication security can be analyzed easily.
- ACORN is a sequential authenticated cipher, and one bit of message is processed in one step. This feature benefits light-weight hardware implementation, and the control circuit in the hardware implementation can be greatly simplified.
- ACORN allows parallel computation. In ACORN, 32 steps can be computed in parallel. This parallel feature benefits high speed hardware and software implementation.
- Length information of associated data and plaintext/ciphertext is not needed in ACORN, i.e., ACORN does not need to check the length of message, and ACORN does not need to pad the message to a multiple of block size (the length of the bits being padded is fixed in ACORN). This feature reduces further the cost of hardware implementation.
- Efficient in Hardware.
  According to our estimation, the hardware cost of ACORN-128 is slightly higher than that of TRIVIUM [6], which is very efficient in hardware.
- Efficient in Software
  In ACORN, 32 steps can be computed in parallel, so its software speed is reasonably fast.
- ACORN has several advantages over AES-GCM: ACORN is more hardware efficient than AES-GCM (especially for hardware resource and energy consumption). On the general computing devices (no AES-NI and no polynomial computing circuits), ACORN is more efficient than AES-GCM in software. The code size of ACORN is very small.

Chapter 5

The Performance of ACORN

Hardware Performance. ACORN is a bit-wise cipher, and it uses very simple feedback circuits and control circuits, so we expect that ACORN is very light-weight in hardware. We compare ACORN with TRIVIUM [6], a well-known hardware efficient stream cipher: the state size of ACORN (293 bits) is very close to that of TRIVIUM (288 bits), and the feedback circuits of ACORN are slighly more complicated than that of TRIVIUM, so we expect the implementation cost of ACORN is very close to that of TRIVIUM.

Note that 32 steps of ACORN can be implemented in parallel in hardware, so we expect that the speed of ACORN is very fast in hardware.

Software Performance. We implemented ACORN in C code. We tested the speed on Intel Core i5-2540M 2.6GHz processor (Sandy Bridge) running 64-bit Ubuntu 11.04 and turning off the Turbo Boost. The compiler being used is gcc 4.5.2, and the optimization option "-O3" is used. In our test, associated data is not considered, and 128-bit tag is used. The test is performed by processing a message repeatedly and printing out the final message.

Table 5.1: The speed (cpb) of ACORN for different message length on Intel Sandy Bridge processor

64B	128B	256B	512B	1024B	2048B	4096B
72.1	41.5	26.3	18.6	14.7	12.8	11.9

Chapter 6

Design Rationale

ACORN is designed to be efficient in hardware (focus), and also efficient in software.

In order to be efficient in hardware, we use a bit-wise stream cipher for its well-known hardware efficiency (such as A5/1 [1], Grain [9] and Trivium [6]). In order to resist the traditional attacks (correlation attacks [13, 12, 10, 11, 3] and algebraic attacks [4, 5]) on stream cipher, the state is updated in a nonlinear way, and every state bit affects the whole state.

We inject message into the state so that we could obtain authentication security almost for free. The challenge is that in a bit-wise stream cipher based on nonlinear feedback registers, it is tremendously difficult to trace the differential propagation in the state, especially if we want to achieve high authentication security (such as 128-bit). Our design focus is to solve this problem so that the authentication security could be easily analyzed. Our solution is to use the concatenation of several linear feedback shift registers to ensure that once there is difference in the state, the number of difference bits in the state would be sufficiently large before the difference gets eliminated. When there are difference bits in the state, the nonlinear function FBK introduces difference noise to the feedback bit f_i so as to reduce the success rate of forgery attack. If an attacker intends to modify the ciphertext, the difference in the keystream bits would also affect the state through the decrypted plaintext bits.

In order to further reduce the hardware complexity, ACORN does not check the message length in decryption and verification, and in ACORN, the padding bits (both length and values) are always fixed. In order to separate the processing of associated data and plaintext/ciphertext, the cipher feedback is modified for 256 steps (through modifying ca_i) before the plaintext/ciphertext gets processed. Similarly, in order to separate the processing of plaintext/ciphertext and the finalization, the cipher feedback is modified (through modifying ca_i) for 256 bits before the finalization.

In order to be fast in hardware and software, 32 steps of ACORN can be computed in parallel.

In order to have high security for stream cipher, when we select the tap

positions, we try to have tap distances which are prime or contain some large prime factor.

In order to resist the differential attack against ACORN for fixed IV (such as the differential attack against Phelix [17]), we require that each key/IV pair is used to protect only one message, and the decrypted plaintext should not be disclosed if the verification fails.

Chapter 7

No Hidden Weakness

We state here that the designer/designers have not hidden any weaknesses in this cipher.

Chapter 8

Intellectual property

We state that ACORN is not patented and it is freely available for all applications.

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

Chapter 9

Consent

The submitter hereby consents 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 understands 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 understands 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 acknowledges that the committee decisions reflect the collective expert judgments of the committee members and are not subject to appeal. The submitter understands that if he disagrees with published analyses then he is expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions. The submitter understands that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

Chapter 10

Changes

We made the following tweaks in the second round submission:

1. The number of steps in the initialization, padding of associated data, padding of plaintext, and finalization are changed from 1536, 512, 512, 512 to 1792, 256, 256, 768, respectively.
   The main reason from the change is to increase the steps in the initialization, so as to provide better protection of the secret key when nonce is reused.

2. In the initialization stage, the key bits are now used as inputs in 1664 steps. (In version 1, the key bits are used only in 128 steps.) The reason for this tweak is to strengthen the cipher against the nonce reuse attack (in encryption/decryption) so that the secret key cannot be easily recovered in the nonce reuse attack.

Bibliography

[1] A5/1. Available at http://en.wikipedia.org/wiki/A5/1

[2] C. Chaigneau, T. Fuhr and H. Gilbert. " Full key-recovery on ACORN in nonce-reuse and decryption-misuse settings." In CAESAR mailing list.

[3] V.V. Chepyzhov, T. Johansson and B. Smeets. "A Simple Algorithm for Fast Correlation Attacks on Stream Ciphers." In *Fast Software Encryption – FSE 2000*, pp. 181-195.

[4] N. T. Courtois. "Fast Algebraic Attacks on Stream Ciphers with Linear Feedback." In *Advances in Cryptology – CRYPTO 2003*, LNCS 2729, pp. 176-194.

[5] N. T. Courtois and W. Meier. "Algebraic Attacks on Stream Ciphers with Linear Feedback." In *Advances in Cryptology – EUROCRYPT 2003*, pp. 345-359.

[6] C. De Canniere, Bart Preneel. Trivium. In *New Stream Cipher Designs – The eSTREAM Finalists*, Springer-Verlag, 2008.

[7] I. Dinur, A. Shamir. Cube Attacks on Tweakable Black Box Polynomials. In *Advances in Cryptology – EUROCRYPT 2009*, pp. 278–299.

[8] I. Dinur, A. Shamir. Breaking Grain-128 with Dynamic Cube Attacks. In *Fast Software Encryption – FSE 2011*, pp. 167–187.

[9] M. Hell, T. Johansson, A. Maximov and W. Meier. "The Grain Family of Stream Ciphers." In *New Stream Cipher Designs – The eSTREAM Finalists*, Springer-Verlag, 2008.

[10] T. Johansson and F. Jönsson. "Improved Fast Correlation Attacks on Stream Ciphers via Convolutional Codes." In *Advances in Cryptology – EUROCRYPT'99*, LNCS 1592, pp. 347-362, Springer-Verlag, 1999.

[11] T. Johansson and F. Jönsson. "Fast Correlation Attacks Based on Turbo Code Techniques." In *Advances in Cryptology – CRYPTO'99*, LNCS 1666, pp. 181-197, Springer-Verlag, 1999.

[12] W. Meier and O. Staffelbach. "Fast Correlation Attacks on Certain Stream Ciphers." *Journal of Cryptography*, 1(3):159-176, 1989.

[13] T. Siegenthaler. "Correlation-Immunity of Nonlinear Combining Functions for Cryptographic Applications." *IEEE Transactions on Information Theory*, IT-30:776-780,1984.

[14] H. Wu and B. Preneel. "Cryptanalysis of the Stream Cipher DECIM." In *Fast Software Encryption – FSE 2006*, LNCS 4047, pp. 30-40, Springer-Verlag, 2006.

[15] H. Wu and B. Preneel. "Resynchronization Attacks on WG and LEX." In *Fast Software Encryption – FSE 2006*, LNCS 4047, pp. 422-432, Springer-Verlag, 2006.

[16] H. Wu and B. Preneel. "Differential attacks on Stream Ciphers Py, Py6 and Pypy." In *Advances in Cryptology – Eurocrypt 2007*, pp. 276-290, Springer-Verlag.

[17] H. Wu and B. Preneel. "Differential attacks on Stream Ciphers Phelix." In *Fast Software Encryption – FSE 2007*, pp. 87-100.