World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:5, 2007

ASC – A Stream Cipher with Built–In MAC Functionality

Kai-Thorsten Wirt

Abstract**—** In this paper we present the design of a new encryption scheme. The scheme we propose is a very flexible encryption and authentication primitive. We build this scheme on two relatively new design principles: t-functions and fast pseudo hadamard transforms. We recapitulate the theory behind these principles and analyze their security properties and efficiency. In more detail we propose a stream cipher which outputs a message authentication tag along with the encrypted data stream with only little overhead. Moreover we propose security-speed tradeoffs. Our scheme is faster than other comparable t-function based designs while offering the same security level.

Keywords**—** Cryptography, Combined Primitives, Stream Cipher, MAC, T-Function, FPHT

I. Introduction

Combined primitives which perform encryption and authentication at the same time are a very interesting design because of the high performance they offer. This is due to the fact, that the data has to be only parsed once and not twice for encryption and authentication. There are different construction principles like dedicated combined primitives (e.g. *Phelix* [25], *VMPC* [26], etc.) which are designed to calculate an authentication tag while performing encryption, or authenticated encryption modes (e.g. *CCM* [24], *GCM* [19], etc.) which employ a block cipher in such a way, that the encrypted data can as well be authenticated. Another possible design are streaming macs like *Mundja* [10]. Streaming macs use input from a stream cipher which is used to encrypt the data, to reduce the costs for the mac computation. This input can be part of the key stream, or of the internal state of the key stream generator. One possibility to construct streaming macs is to modify hash functions such that they process the message along with the secret input from the stream cipher.

In this paper we present a combined primitive, which is based on a stream cipher together with the streaming mac idea. We call this primitive *ASC*. Our goal is to combine different cryptographic primitives which are based on theoretical considerations into a practical stream cipher with authentication ability. Our motivation is to present a practical implementation of theoretic constructions in order to show, that these abstract primitives can be used in practice.

The underlying stream cipher of our combined primitive is based on *t-functions*. T-functions are a relatively new design principle proposed by Klimov and Shamir [14], [15], [16], [13] and Anashin [1], [2]. Moreover there are various new stream cipher designs based on t-functions [3], [18], [11]. The major advantages of t-functions in stream cipher designs are their efficiency, their desireable properties like large cycle lengths and their security against algebraic attacks.

[1]Kai Wirt is with Technische Universität Darmstadt, Germany

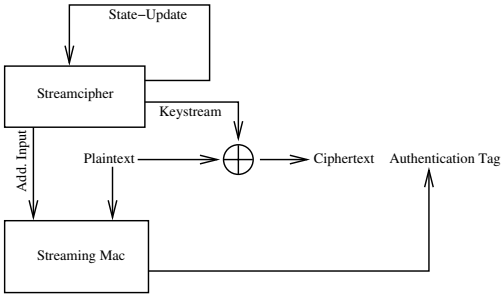

Fig. 1. Streaming Mac

The streaming mac of our design is based on the FPHT hash [7]. This hash function uses fast pseudo-hadamard transforms as diffusion layer along with the AES S-Box for confusion.

ASC in the basic version offers a comparable speed than other t-function based designs. However we also present a speed-optimized version which runs faster. Our primitive proposes a new way, how stream ciphers and hash functions can be combined to offer authenticated encryption in one primitive.

ASC has an internal state of 256 bits which is made up of 8 32-bit words. The cycle length of ASC is therefore 2^{256}. ASC outputs one 32-bit word each clock and the MAC size is 128 bits.

In the following we first present the streaming mac design. In Section III we present the FPHT and our mac generation algorithm. Section IV describes the idea behind multi-word t-functions and our stream cipher approach. The concrete design of ASC is the topic of Section V. A security analysis is given in Section VI and the conclusion is presented in Section VII.

II. Streaming Mac

Streaming macs make use of an *authentication* tag generation algorithm and a stream cipher. The stream cipher is used to encrypt the data and to generate an additional input which is fed into the tag generation algorithm. This algorithm is basically a mac function, which takes the additional input along with the message and generates the authentication tag. The idea behind streaming macs is, that the data from the stream cipher can be used to reduce the costs for the mac computation. The basic principle of the streaming mac design is shown in Figure 1.

The underlying stream cipher of our design is based on a t-function proposal presented in [15]. This stream cipher outputs one part of the internal state for encryption and another part for the input into the mac function. Additionally we modify

the primitive in order to enhance inter state mixing and we add a key scheduling procedure to initialize the internal state.

For the mac function of our design we modify the FPHT hash function [7]. We reduce the number of rounds and the authentication tag length in order to be able to speed up the tag calculation. In order to strengthen this variant we modify the compression function such that it uses the additional input from the stream cipher. In principle we hold on to the design of FPHT hash in order to adopt the security claims for our mac function.

III. The FPHT MAC

Our authentication tag is calculated using a variant of the hash function presented in [7]. We first give a short introduction on the Fast Pseudo Hadamard Transform.

Definition 1 (Fast Pseudo Hadamard Transform): We use the FPHT as linear layer in our design. This transform can recursively be described by

$$H_0 = 1$$
$$H_n = \begin{bmatrix} 2 \cdot H_{n-1} & H_{n-1} \\ H_{n-1} & H_{n-1} \end{bmatrix} \quad \text{for } n \geq 1$$

The quality of the diffusion effect of a linear layer is meassured using the *Branch number* [6]. In principle the Branch number indicates how the modification of one bit in one round affects the state of the next round. The higher the Branch number, the more security can be achieved using the linear layer. The Branch number for an n-bit state is bounded by $n + 1$. This value is reached using *Maximum-Distance Separable (MDS)-Codes*. Although the FPHT does not offer the Branch number of MDS codes it has some nice implementation benefits. Because of the recursive structure the FPHT is faster than MDS calculation. While MDS codes have complexity $O(n^2)$ the FPHT can be computed in $O(n \log n)$. Since the FPHT requires only one multiplication in the recursion it is possible to precompute multiplication tables. For instance the H_2 can be implemented efficiently by using tables for the multiplication with x and x^2 and some logical operations. And the H_4 can be calculated by applying the H_2 twice. In addition to the efficient implementation, the diffusion effect of the fast pseudo hadamard transform is good, albeit not optimal. The exact branch number for the H_4 is 8 and for the H_5 is 12 [7]. Since the FPHT has several efficient means of implementation the design is very flexible and allows for fast implementations on various platforms.

A. Modifications to FPHT Hash

The FPHT hash computes message digests of 256 bits. We reduce the size of the internal state to 128 bits. Thus we can reduce the number of rounds from 6 to 4. This results in a faster tag calculation however it weakens the mac function. Thus we have to improve the security of the single rounds. To achieve this, we replace the known AES S-Boxes by unknown random S-Boxes. Since random S-Boxes don't have optimized properties like the AES S-Box they have to be bigger in order to prevent differential or linear attacks. We have chosen 32 bit S-Boxes because this is still the word size of common processors. Thus operations on 32-bit words can be carried out efficiently. Since the S-Boxes are generated by the used stream cipher we suppose that the used stream cipher can efficiently compute the needed 32 bit words without much overhead compared to the encryption. Security considerations regarding 32 bit random S-Boxes are presented in Section VI on page 5. As a short note, the probability for differentials or linear approximations decreases very fast with the number of bits in the S-Box. Therefore it is very unlikely that good approximations which can be exploited exist for 32 bit S-Boxes.

Since we have to adopt the linear layer to the new state size of 128 bits as well, we can replace the H_5 by the H_4 transform. This shortens the calculation of the FPHT by one recursion level thus speeding up the mac computation additionaly.

The last modification to FPHT hash is the message expansion i.e. the key schedule of the hash function. Since we now have only 4 rounds, the key schedule can be shortened as well. We point out, that since the key schedule of FPHT hash is designed to dwarf attacks already in the first round and the first round keys are already calculated in a secure way the shortened key schedule is also strong against linear and differential attacks.

B. The Algorithm

1) Preparation: FPHT streaming mac takes inputs in blocks of 512-bits and produces macs of 128-bits. The input message is padded by a single one bit followed by enough zeros to make the length congruent 448 modulo 512. The length of the original message is appended in 64-bit representation. Every block of the message is then compressed using the function described below. All values are loaded and stored in little endian byte order.

2) Compression Function: In the following we describe our compression function. It takes a message block $T_{0\ldots15}$ and an internal state $S_{0\ldots3}$ which are both arrays of 32-bit words. The state is initialized with $S_i = 286331153 \cdot i$. Moreover it takes the additional input from the partner stream cipher A which is a 32-bit word. This input has to be updated for every message block.

In Algorithm 1 the H_4 is calculated over the field $GF(2)[x]/(x^8 + x^4 + x^3 + x^2 + 1)$. All operations are on 32 bit words, i.e. multiplication and addition are performed modulo 2^{32}. The notation $\lll 11$ indicates a left cyclic shift by eleven bits.

The message expansion in lines 7 and 8 is essentially the same as the one of FPHT hash. The only difference is the number of iterations since we now only have a state size of 128 bits. In the compression function in lines 10 through 15 we have reduced the number of rounds to 4. Moreover we have replaced the AES S-Box by the multiplication with the additional input from the stream cipher.

C. Implementation Issues

1) FPHT: As pointed out in [7] the fast pseudo-hadamard transform can be implemented very efficiently. In hardware it is sufficient to implement the H_1 directly, which requires

Algorithm 1: FPHT Streaming Mac Compression Function

```
1  for x = 0 to 3 do
2  |  L_x ← S_x;
3  end
4  for x = 0 to 15 do
5  |  W_x ← T_x;
6  end
7  for x = 16 to 31 do
8  |  W_x ← (W_{x-16} ⊕ W_{x-14} ⊕ W_{x-8} ⊕ W_{x-3} ⊕ x) ⋘ 11;
9  end
10 for x = 0 to 3 do
11 |  for y = 0 to 3 do
12 |  |  L_y ← (L_y ⊕ W_{16+4x+y}) · A;
13 |  end
14 |  L ← H_4(L);
15 end
16 for x = 0 to 3 do
17 |  S_x ← S_x + L_x;
18 end
```

only a simple multiplication by $p(x) = x$. Higher-dimensional transforms can then be implemented using only additional xor-gates.

In our implementation we have chosen to implement multiplication tables for the multiplication with $p(x) = x$ and $p(x) = x^2$. Using these tables we can realize the H_2 transform using 48 xor-transformations. The required H_4 is then basically twice the H_2 and can therefore be implemented using 96 xor operations and 56 table lookups.

A full table driven implementation would require the precomputation of 16 tables. This method would need 16 table lookups and 16 xor operations to calculate the H_4. Our approach is a tradeoff between storage and efficiency. We want to stress, that the fast pseudo hadamard transform scales better than equal dimensional MDS-codes. Moreover for the FPHT there are more optimizations available than for MDS codes.

2) S-Box: The S-Box of the FPHT streaming mac is the multiplication with the additional input from the stream cipher. Since the same value is used for a complete block the stream cipher and the FPHT streaming mac can be implemented in parallel. While the mac is calculated for one block the stream cipher can encrypt that block at the same time. Moreover various 32-bit multiplications can be carried out in parallel on many modern processors. Therefore it is often possible to calculate 2 or even all 4 S-Box applications in line 12 in Algorithm 1 in one operation.

IV. Multi-Word T-Functions

In the design of stream ciphers there are mainly two approaches. The first one is to build simple and good to analyze stream ciphers like for example ones using *linear feedback shift registers*. While certain properties like cycle length and statistics can be easily shown to meet some design criteria this simplicity may also lead to possible attacks. The other extreme are stream ciphers which are too complex to analyze. The designer hopes, that the cipher is strong enough while the attacker has the problem that the cipher offers no obvious attacking points.

A few years ago, Klimov and Shamir [14], [15], [16], [13] and Anashin [1], [2] proposed a compromise between simplicity and complexity which they called *t-functions*. T-functions are mappings where the i-th bit of the output depends only on bits 0 through i of the input. Klimov, Shamir and Anashin gave criteria for which t-functions are single- cycle permutations. That is the repeated application of an n-bit t-function runs through all possible values $0 \dots 2^n$. Klimov and Shamir proposed one example of a single cycle t-function which uses only three instructions. This mapping $f(x) = x + (x^2 \vee 5)$ serves as basic mapping in our design.

One drawback with t-functions is, that they are only efficient when the word size is the word size of the underlying processor since only in this case the function can be evaluated using simple machine instructions. One possible solution is to use t-functions only as part in a stream cipher construction and not as (main)keystream generator itself (e.g. [3]).

Another solution are *multi-word* t-functions as proposed by Klimov and Shamir. They showed how one can use k n-bit functions to construct functions with $2^{k \cdot n}$ cycle length. Using this technique it is possible to use simple instructions to calculate each of the k functions separately or even in parallel. In [16] Klimov and Shamir propose several possible constructions. In this paper we present a design based on one of these ideas. Our design uses 8 32-bit t-functions which sums up to an internal state size of 256 bits.

The basic idea behind the multi-word construction we use is that of a counter where the single t-functions can be seen as digits. The basic version developed in [16] is

$$f_i(x_0, \dots, x_k) = x_i \oplus (\alpha_i(x_0, \dots, x_{k-1}) \wedge x_0 \wedge \dots \wedge x_{i-1})$$

where each α_i is an odd parameter, that is

Definition 2 (Odd Parameter):

$$\bigoplus_{(x_0,\dots,x_{k-1})=(0,\dots,0)}^{2^n-1,\dots,2^n-1} [\alpha_i(x_0, \dots x_{k-1})]_n = 1 \qquad (1)$$

and $[\alpha_i]_0 = 1$. The notation $[x]_i$ denotes the i-th bit of x. The equation f is the formalization of the concept of a counter. Each register x_i is updated using α_i if the previous registers $x_{i-1} \dots x_0$ all have the same value. This means register x_0 is updated every clock, register x_1 is updated every second clock, register x_2 is updated every 4-th clock and so on.
For α_i we use the function

$$H(x) = (x + (x^2 \vee 5)) \oplus x$$

which is an odd parameter. This function is a single cycle t-function which can be realized with a minimal number of operations.Using H our first approach for a multi-word t-function becomes

$$f_i(x_0, \dots, x_k) = x_i \oplus (H(x_0 \wedge \dots \wedge x_{k-1}) \wedge x_0 \wedge \dots \wedge x_{i-1})$$

using the counter technique mentioned above. The drawback with this simple approach is that the inter-word mixing is

not good. More formally in this mapping each bit of x_{k-1} is changed with probability $2^{-(k+1)}$ instead of $\frac{1}{2}$ according to [16].

This problem can be solved using an additional parameter. It is possible to add an even parameter with zero in the least significant bit without modifying the cycle length.

Definition 3 (Even Parameter): A parameter r is called even, if

$$B[r,n] := 2^{-n} \sum_{i=0}^{2^{n-1}-1} (r(i+2^{n-1}) - r(i))(\mathrm{mod}2)$$

is always zero[1].
In our case we use the parameter given by

$$g_i(x_0 \dots x_k) = \begin{cases} 2 \cdot x_{i-1} \cdot x_{i-2} & i > 1 \\ 2 \cdot x_k \cdot x_0 & i = 1 \\ 2 \cdot x_{k-1} \cdot x_k & i = 0 \end{cases}$$

This parameter is used to speed up the inter-register mixing. Since it uses each two adjacent registers which make up the internal state, a modification in one bit of the state evolves through all registers after at most k rounds. Thus states where the modifications to the state induced by the t-function are only small are left very fast. Using this parameter the complete multiword t-function used in our design is given by

Definition 4 (Round Function of ASC):

$$f_i(x_0, \dots, x_k) = x_i \oplus (H(x_0 \wedge \dots \wedge x_{k-1}) \wedge x_0 \wedge \dots \wedge x_{i-1}) \oplus g_i(x_0 \dots x_k)$$

V. Design

In our design we have chosen to use 32-bit words, since this is still a widely used word size on common microprocessors. Moreover processors having larger word sizes are normally able to carry out several 32-bit operations in parallel thus speeding up an implementation of ASC.

We use 8 32-bit words numbered from $i = 0$ to $i = k = 7$. Each clock cycle the state is updated according to equation 2 in Section IV. The cipher text is calculated by xoring the plain text with the register $i = 7$ of ASC.

After each encryption of a multiple of 512 bits the state is updated once more. After this update the content of register $i = 6$ i.e. the second last register is fed into the FPHT streaming mac (see Section III on page 2) as additional input and the mac state is updated for the last 512 bit block. Note, that the mac is computed over the plain text, not the produced cipher text.

After the encryption of the last plain text words the plain text is padded according to Section III-B.1 on page 2. However the padded bits are not encrypted since this is not necessary for stream ciphers of course. As with the normal encryption the state is updated once more before feeding $i = 6$ into the macfunction and calculating the mac for the last block.

[1]In the multivariate case every parameter which does not use all the variables is even. [16]

A. Key Schedule

We want to keep the Key Schedule of ASC rather simple. However it is not possible to use the state update function itself because of the nature of t-functions. Doing so higher order bits would only have little influence on the starting state. For the Key Schedule we add the parameter

$$h_i(x_0 \dots x_k) = \frac{1}{2} \cdot x_i$$

i.e. the current register is shifted to the right by one bit. Therefore higher order bits propagate to the right due to the parameter h, while lower order bits propagate to the left, due to the remaining t-function. Note, that the complete Key Scheduling Function is not a t-function anymore, therefore we can not make any statements on the cycle structure of the Key Schedule. However we believe, that given the large state it is very unlikely that too much entropy is lost during key setup. To produce the starting state, the Key Bits are loaded into the registers starting with register 0 in little endian order in 32-bit blocks. If the key is shorter than 256 bits the remaining registers are initialized with 0, however we do not recommend keys shorter than 128 bits. After that initial step the modified update function

Definition 5 (Key Scheduling):

$$f'_i(x_0, \dots, x_k) = x_i \oplus (H(x_0 \wedge \dots \wedge x_{k-1}) \wedge x_0 \wedge \dots \wedge x_{i-1}) \oplus g_i(x_0 \dots x_k) \oplus h_i(x_0 \dots x_k)$$

is executed 256 times to calculate the starting state. Of course no output is produced during the Key Schedule.

B. Security-Speed Trade-Off

We have been very conservative in the number of bits which are produced by ASC per cycle. However it is possible to speed the key stream generation up by reducing the security level. ASC can output more than just one register. It is possible for example to output registers 5,6 and 7 and use register 4 as input to the macfunction. This way ASC would produce 96 bits output per clock cycle. Clearly this reduces the number of unknown bits to 160. We stress that not more than 128 bits i.e. four registers should be output per clock cycle in order to avoid exhaustive search attacks.

C. Performance

Our unoptimized implementation of ASC encrypts at 72 clock cycles per Byte. We have compared the performance to other t-function based designs. *Mir-1* [18] which is a similar design runs at about 39 clocks per byte. However the data has to additionally be authenticated using some mac algorithm. *TSC-3* [11] runs at about 50 clocks per byte, again without authentication. Thus if the mac computation is slower than 33 clocks per byte, the unoptimized version of ASC is faster than these two t-function based designs.

If we use the security-speed tradeoff mentioned above and output 96 bits per clock cycle, ASC encrypts at about 24 clocks per byte. This is even faster than the above mentioned ciphers which don't have an authentication ability. Compared to Phelix, another combined primitive, this is slightly slower.

TABLE I
PERFORMANCE COMPARISON

Algorithm	Clock Cycles per Byte[2]	Authentication
ASC standard	72	yes
TSC-3	50	no
MIR-1	39	no
ASC 96 bits output	24	yes
Phelix	14	yes

Phelix [25] encrypts at 14 clock cycles per byte with an optimized C implementation. However we feel, that an optimized C implementation of ASC gains some more speed.

This performance comparison is summarized in Table I. We want to point out that the complete t-function based ASC cipher is even faster than MIR-1, where the t-function is only used to enhance the security of a linear feedback shift register based design.

VI. Security Analysis

A. The Streaming Mac

In order to be able to perform a security analysis of the FPHT mac we have to make the assumption that the used stream cipher is secure. It is obvious, that the security of the mac depends strongly on the randomness of the stream cipher input since this is used to generate the S-Boxes. Therefore we stress, that FPHT mac should only be used with stream ciphers, which offer or exceed the required security level. The security of the stream cipher used together with FPHT mac in ASC is analyzed in Section VI-B.

1) Key Schedule: The key schedule of our FPHT streaming mac is essentially the same as the one of the FPHT hash function. Therefore the security claims in [7] also hold for our design. The only difference is that the key schedule is shorter. However the FPHT hash key schedule was designed to use all of the input already in the first round. Moreover the input key itself is not used in the round function. All the round keys are generated by the linear mixing in the key schedule. Therefore we believe that the key schedule of the FPHT streaming mac is secure against related key attacks.

Additionally the key schedule is designed to make it harder to find differential trails. Since the initial input state is fixed the attacker could create a differential trail of probability one through rounds where the input key itself is used. As pointed out above, the round keys are a function of the input. Therefore the probability one differential is harder to ensure. This is already true for the first round keys and therefore it is also true for the shortened key schedule.

The argument, that simple slide attacks are prevented by using a linear mixing with the key word number holds for the shortened key schedule as well.

2) The Compression Function: If we assume the stream cipher to be secure, this means, that the S-Boxes used in FPHT mac are chosen at random from the set of all possible multiplicative S-Boxes for each input block. It is well known, that small random S-Boxes are not resistant against differential cryptanalysis [4] [5]. However the probability for the existence of high differential characteristics or linear expressions for output bits is decreasing very fast with the size of the S-Box [21]. For example no output bit of a 32-bit S-Box can be described as a linear function of the input bits with high probability [9]. Moreover random S-Boxes offer resistance against linear cryptanalysis after a certain number of rounds [12]. Together with the H_4 which has branch number 8 the number of active S-Boxes must be at least 16 over four rounds. Therefore we assume, that our compression function is resistant against differential or linear cryptanalysis.

Additionally random S-Boxes have one advantage over well chosen ones. They may not be as resistant to known attacks as selected S-Boxes, but are more likely to be resistant to unknown attacks. Also since the S-Boxes are changed based on the encryption key they are different for every message block. Therefore conventional attacks only work for one block and are not applicable to multi-block messages in a straightforward manner. Instead new techniques involving the S-Box schedules have to be developed. We feel, that it is difficult to even apply standard differential or linear cryptanalysis to the compression function of FPHT streaming mac as far as more than one message block is analyzed within an attack. In the scope of ASC this effect is even greater, because the stream cipher is updated more than once before another input for the mac function is produced. Therefore it is hard to express the S-Box for the next message block using the S-Box for previous ones.

B. The Stream Cipher

1) Key Schedule: We decided to use a modified update function in the key setup a number of times. Thus we believe that the Key Scheduling does not introduce weaknesses in the design of the streamcipher. Moreover we believe that because of the non-linear update function it becomes difficult to calculate keys having certain properties. Additionaly since we do not use a t-function per se during the Key Schedule every bit of the key has the same effect on the starting state.

2) Attacks on T-Functions: There have been a number of attacks on t-function based designs for example [17] or [20]. These attacks make use of the fact, that if high order bits of one register are known it becomes possible to calculate the low order bits faster than exhaustive search if a t-function is used in the update. In our case we output only a very conservative number of bits from the state (i.e. 32 of 256). Additionally since we output a complete register and not parts of one the task for the cryptanalyst is different from the above mentioned scenario. Therefore we believe that the so far presented attacks do not apply to ASC.

3) State-Update Function: The state update function of ASC is based on one of the multi-word t-functions presented in [16]. Therefore the registers run through all possible states before repeating. Thus the cycle length of ASC is 2^{256}. However we modified the used parameter to speed up inter-register mixing as described above. This does not shorten the cycle length though. Therefore attacks exploiting short cycle lengths or impossible states are not applicable to ASC.

[2]Performance of ASC was measured on a Mobile Intel Celeron CPU 2.00 GHz, Gentoo Linux 2.6.17 Kernel, GCC 4.1.1. Performance values of the other candidates was taken from the Ecrypt stream cipher project [8] publications

The state update function of ASC is highly nonlinear therefore we believe that simple algebraic attacks don't work. This is also fortified by the fact, that t-functions are developed as a replacement to linear feedback shift registers which are susceptible to algebraic attacks. The nonlinear nature of t-functions generates equations of higher degree with every state update thus the weaknesses of LFSRs are fixed.

Tests with *ENT* [23] and the *NIST* Test Suite [22] showed no weaknesses in the output of ASC. Therefore we believe, that general attacks exploiting the statistics of the generated key stream do not work against ASC. We feel, that encrypted data can not be distinguished from random noise. Also we are not aware of any distinguisher for the used t-function construction. Additionaly we believe, that our modified parameter does not substantialy weaken the state-update function therefore the security arguments for multi-word t-functions given in [16] hold for our construction as well.

VII. Conclusions

In this paper we have presented a new combined primitive, which offers authentication and encryption at the same time. Our design uses the streaming mac idea which uses the encryption function to reduce the costs for the authentication procedure. The primitives we used are relatively new and promising and thus are interesting principles for further research in this area. Our goal was to combine these two building blocks in a concrete cipher design.

In this paper we have explained the interaction between mac function and stream cipher in the streaming mac design. The combination of encryption and authentication ability allows for more efficient authenticated encryption than the separate application of the different primitives would. We have presented the basic building blocks for the streaming mac design and proposed a concrete example which we called ASC. We have modified two existing abstract primitives and developed practical variants which can be put together to produce a streaming mac function.

Additionally we have developed a practical implementation of an abstract cryptographic primitive namely a t-function based stream cipher. We have fixed a secure parameter set for this stream cipher and have added necessary functionality like key scheduling and initialization. Moreover we have argued that existing attacks on t-function based designs do not apply to our approach because of the different keystream generation method.

Finally we have presented a security-speed tradeoff for our stream cipher which can increase the performance of ASC significantly. Additionally we have given performance meassures and security claims for our new stream cipher.

References

[1] Vladimir Anashin. Uniformly distributed sequences of p-adic integers, ii. arXiv Mathematics, 2002. http://arxiv.org/abs/math.NT/0209407.

[2] Vladimir Anashin. Pseudorandom number generation by p-adic ergodic transformations. arXiv Mathematics, 2004. http://arxiv.org/abs/cs.CR/0401030.

[3] Vladimir Anashin, Andrey Bogdanov, Ilya Kizhvatov, and Sandeep Kumar. Abc : A new fast flexible stream cipher. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/001, 2005. http://www.ecrypt.eu.org/stream.

[4] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In *Proceedings of CRYPTO 1990*, volume 537 of *Lecture Notes in Computer Science*. Springer Verlag, 1990.

[5] Eli Biham and Adi Shamir. Differential cryptanalysis of snefru, khafre, redoc-ii, loki and lucifer. In *Proceedings of CRYPTO 1991*, volume 576 of *Lecture Notes in Computer Science*. Springer Verlag, 1991.

[6] Joan Daemen. *Cipher and hash function design: strategies based on linear and differential cryptanalysis*. PhD thesis, Katholieke Universiteit Leuven, 1995.

[7] Tom St Denis. Fast pseudo-hadamard transforms. Cryptology ePrint Archive, Report 2004/010, 2004. http://eprint.iacr.org/.

[8] ECRYPT. estream, the ecrypt stream cipher project, 2004. http://www.ecrypt.eu.org/stream/index.html.

[9] J. A. Gordon and H. Retkin. Are big s-boxes best? In *Proceedings of the Workshop on Cryptography*, volume 149 of *Lecture Notes in Computer Science*. Springer Verlag, 1982.

[10] Philip Hawkes, Michael Paddon, and Gregory G. Rose. The mundja streaming mac. Cryptology ePrint Archive, Report 2004/271, 2004. http://eprint.iacr.org/.

[11] Jin Hong, Dong Hoon Lee, Yongjin Yeom, Daewan Han, and Seongtaek Chee. T-function based stream cipher tsc-3. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/031, 2005. http://www.ecrypt.eu.org/stream.

[12] Liam Keliher. *Linear Cryptanalysis of Substitution-Permutation Networks*. PhD thesis, Queen's University, Kingston, Canada, 2003.

[13] Alexander Klimov. *Applications of T-Functions in Cryptography*. PhD thesis, The Weizmann Institute of Science, 2005.

[14] Alexander Klimov and Adi Shamir. A new class of invertible mappings. In *Proceedings of CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*. Springer Verlag, 2002.

[15] Alexander Klimov and Adi Shamir. Cryptographic applications of t-functions. In *Proceedings of SAC 2003*, volume 3006 of *Lecture Notes in Computer Science*. Springer Verlag, 2003.

[16] Alexander Klimov and Adi Shamir. New cryptographic primitives based on multiword t-functions. In *Proceedings of FSE 2004*, volume 3017 of *Lecture Notes in Computer Science*. Springer Verlag, 2004.

[17] Simon Künzli, Pascal Junod, and Willi Meier. Distinguishing attacks on t-functions. In *Proceedings of Mycrypt 2005*, volume 3715 of *Lecture Notes in Computer Science*. Springer Verlag, 2005.

[18] Alexander Maximov. A new stream cipher mir-1. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/017, 2005. http://www.ecrypt.eu.org/stream.

[19] David A. McGrew and John Viega. The galois/counter mode of operation (gcm). Submission to NIST Modes of Operation Process, 2004. http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/.

[20] Joydip Mitra and Palash Sarkar. Time-memory trade-off attacks on multiplications and t-functions. In *Proceedings of ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*. Springer Verlag, 2004.

[21] Luke O'Connor. On the distribution of characteristics in bijective mappings. In *Proceedings of EUROCRYPT 1993*, volume 765 of *Lecture Notes in Computer Science*. Springer Verlag, 1993.

[22] Andrew Rukhin, Juan Soto, and James Nechvatal et al. A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications, 1997. http://csrc.nist.gov/rng/.

[23] John Walker. Ent – entropy calculation and analysis of putative random sequences, 1985. http://www.fourmilab.ch/random/.

[24] Doug Whiting, Russ Housley, and Niels Ferguson. Counter with cbc-mac (ccm). Submission to NIST Modes of Operation Process, 2004. http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/ccm/.

[25] Doug Whiting, Bruce Schneier, Stephan Lucks, and Frederic Muller. Phelix - fast encryption and authentication in a single cryptographic primitive. Ecrypt Stream Cipher Project, Report 2005/020, 2005. http://www.ecrypt.eu.org/stream.

[26] Bartosz Zoltak. Vmpc-mac: A stream cipher based authenticated encryption scheme. Cryptology ePrint Archive, Report 2004/301, 2004. http://eprint.iacr.org/.