Customizable Sponge-Based Authenticated Encryption Using 16-bit S-boxes

Matthew Kelly[1], Alan Kaminsky[2], Marcin Łukowiak[1],
Michael Kurdziel[3], and Stanisław Radziszowski[2]

[1]Department of Computer Engineering, Rochester Institute of Technology
[2]Department of Computer Science, Rochester Institute of Technology
[3]Harris Corporation, Rochester, NY

October 2014

Abstract

Authenticated encryption (AE) is a symmetric key cryptographic scheme that aims to provide both confidentiality and data integrity. There are many AE algorithms in existence today. However, they are often far from ideal in terms of efficiency and ease of use. For this reason, there is ongoing effort to develop new AE algorithms that are secure, efficient, and easy to use.

The sponge construction is a relatively new cryptographic primitive that has gained popularity since the sponge-based KECCAK algorithm won the SHA-3 hashing competition. The duplex construction, which is closely related to the sponge, provides promising potential for secure and efficient authenticated encryption.

In this paper we introduce a novel authenticated encryption algorithm based on the duplex construction that is targeted for hardware implementation. We also provide explicit customization guidelines for users who desire unique authenticated encryption solutions within our security margin. Furthermore, our substitution step uses 16×16 AES-like S-boxes, which are novel because they are the largest bijective S-boxes to be used by an encryption scheme in the literature and are still efficiently implementable in both hardware and software.

1 Introduction

The overarching goal of symmetric key cryptography is to enable people to communicate privately over an insecure channel in the presence of adversaries. Two fundamental requirements for achieving this goal are encryption and authentication. Encryption provides *confidentiality* while authentication provides data *integrity* and assurance of message origin [1].

Many authenticated encryption algorithms are in existence today, but they are often unsatisfactory in terms of performance, security, or ease of use. Some algorithms require two passes per block of plaintext to encrypt and authenticate. This is generally undesirable because it often means a much slower algorithm. Other algorithms have been shown to be insecure or difficult to use properly. Many algorithms, such as the ones based on generic composition, require two unique and unrelated keys. This should be avoided when possible because key management is a difficult problem [2].

Furthermore, a new authenticated encryption algorithm is needed that meets the stringent requirements of government and military applications. Such algorithms are not typically in the public domain. The goal of this is partially to reduce or eliminate academic interest in cryptanalyzing the algorithm and publishing results [3].

For this reason, there is a need for a customizable authenticated encryption algorithm. This algorithm should remain secure as long as customizations are made within certain guidelines. The result is an algorithm which can be made unique on a per-user or per-application basis without the effort of cryptanalyzing every specific instantiation. We present a customizable AE algorithm here that is based on the duplex construction and is built around an iterated permutation that uses 16×16 bijective S-boxes. These large S-boxes introduce much higher non-linearity and algebraic complexity to the permutation than what could be achieved with smaller S-boxes of a similar structure (e.g. the AES S-boxes). They are also very efficient to implement in both hardware and software because they are based on algebraic operations in a finite field rather than a random mapping [4].

2 Sponge Construction

The sponge construction is a relatively new cryptographic primitive that has gained popularity since KECCAK won the Secure Hash Algorithm (SHA-3) competition in 2013 [5][6]. Essentially, it provides a way to generalize hash functions (which normally have outputs of fixed length) to functions with arbitrary length output. This generalization allows cryptographic sponges to be used for applications other than hashing.

Sponges are based on the iteration of an underlying function f. This function can either be a general *transformation* or a *permutation*. The security proofs are different for transformations and permutations, and there are advantages and disadvantages for each choice of a function type [7].

The output Z of the parameterized sponge construction is given as

$$Z = \mathbf{sponge}[f, \mathbf{pad}, r](M, \ell),$$

where $\mathbf{pad}$ is a padding function for the input, r is the *rate* of absorption, M is the message (or other input) data, and ℓ is the desired output length. The sponge construction is stateless; there is no information stored between calls to it. It is split into two distinct phases: the *absorbing* phase and the *squeezing* phase. Inputs (e.g. message and/or key material) are absorbed in the first phase and the output (e.g. a MAC or keystream) is squeezed out in the second phase.

The state of the sponge construction is split into two contiguous portions: the *outer state*, which is accessible externally, and the *inner state*, which is hidden. The size of the outer state is given by the *rate* r and the size of the inner state is specified by the *capacity* c. The size of the entire state is $b = r + c$. The speed of the construction depends on the rate, while the security depends on the capacity.

The padding function $\mathbf{pad}$ is first applied to M to make it a multiple of r. M is then absorbed r bits at a time. More concretely, absorption is the process of XORing r-bit blocks into the state while interleaving with applications of the underlying sponge function f. If the rate is increased then more bits are absorbed at a time and thus the construction runs faster. However, increasing the rate means that the capacity must decrease and so there is a clear trade-off between speed and security. Squeezing consists of concatenating r bits at a time to an output bitstring Z that is truncated to ℓ bits. The sponge function f must be called once for each r bits of output after the first full block.

3 Duplex Construction

The duplex construction is highly related to the sponge construction. The main differences are that the duplex construction maintains state between calls and that there no longer exists a clear separation between the absorbing and squeezing phases. Absorbing and squeezing happen essentially at the same time, hence "duplexing" [8]. The duplex mode has several applications [9], with authenticated encryption being the one of obvious interest to us.

Parameters for the duplex construction are mostly the same as for the sponge construction. However, since the duplex construction maintains state, we build a *duplex object* D and make calls to it. The function which processes inputs and produces outputs is called $\mathbf{duplexing}$:

$$Z_i = D.\mathbf{duplexing}(\sigma_i, \ell_i)$$

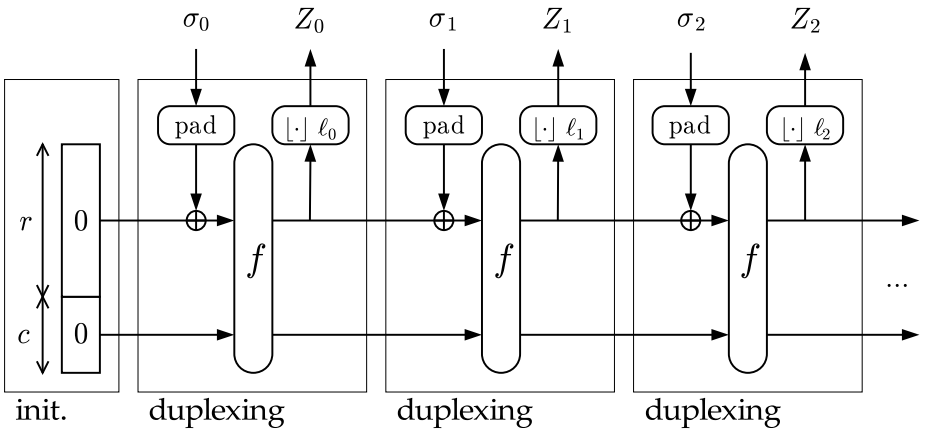

Figure 1: The duplex construction $\mathbf{duplex}[f, \mathbf{pad}, r]$ [7]

Figure 1 shows the duplex construction. The i-th input is denoted σ_i and the i-th output is denoted Z_i, which is truncated to ℓ_i bits. Inputs are absorbed and processed at the same time that outputs are squeezed. For a duplex object it is possible to have an empty input or to not request an output. A *blank call* is a call to $\mathbf{duplexing}$ for which no input is provided ($|\sigma_i| = 0$). A *mute call* is a call for which no output is requested ($\ell_i = 0$).

3.1 Duplex for Authenticated Encryption

Authenticated encryption is easily achieved using the duplex construction. Figure 2 shows such a use case. First, we construct a duplex object D. Then we absorb the key K (or optionally $K||IV$) using one or more mute calls to $D.\mathbf{duplexing}$. More that one mute call may be required if the length of the key exceeds the rate r. We denote a header input to D as A; these arbitrary length inputs are authenticated but not encrypted. We denote a body input to D as B; these arbitrary length inputs are both encrypted and authenticated. A inputs are absorbed using one or more mute calls to $D.\mathbf{duplexing}$. B inputs are absorbed in a similar fashion and then the keystream Z is XORed with B to produce the ciphertext C. The tag T is produced using a blank call to $D.\mathbf{duplexing}$ after all header and body inputs have been processed.

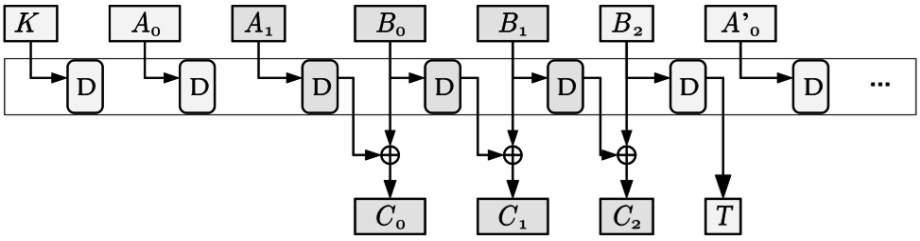

Figure 2: The duplex construction as used for authenticated encryption [9]

We note that the duplex construction may require *domain separation*, a generic mechanism for eliminating output ambiguity. For example, the simplest domain separation method consists of appending a *frame bit* to the last block of every different input data type (e.g. key or message data). This frame bit has the property that no two consecutive data types have the same frame bit value, meaning that one can easily identify where one data type ends and the next begins [8].

3.2 Generic Security

Any calls made to the duplex construction can be reduced to calls to the keyed sponge construction. As a result, the security of the duplex construction depends only on its corresponding sponge construction.

The security of the sponge construction is based on the assumption that the underlying sponge function f is secure. That is, if f is computationally indistinguishable from random then so should be the sponge construction it is instantiated within. Consequently, cryptographers designing a system based on the sponge construction need only be concerned with designing and cryptanalyzing a secure underlying function. The sponge construction, when used properly, is said to be secure against *generic attacks* – attacks which do not exploit any specific properties of the underlying sponge function. We call this the *generic security* of the construction [7].

The generic security of keyed constructions is higher than unkeyed. For our purposes we are interested only in the security of the keyed sponge construction where a permutation is used for f. Jovanovic et. al [10] proved in 2014 that the generic security level of keyed sponge constructions is lower bounded by

$$\min(2^{(r+c)/2}, 2^c, 2^{|K|}).$$

4 Algorithm Specification

Our authenticated encryption algorithm is based on a simplified duplex construction. Padding and domain separation are assumed to be performed at some higher level in the overall system if needed. For this reason, it is sufficient to specify only the duplex parameters and the sponge function f.

4.1 Duplex Parameters

We allow two key sizes: 128 bits and 256 bits. These are the current NIST-recommended values for symmetric key cryptographic primitives [11]. Our construction uses a 512-bit internal state, thus $b = 512$. The rate r is 128 bits for both key lengths, which means that the capacity c is 384. Keeping

the rate at a constant 128 bits for both instantiations means that switching between key lengths is a trivial task.

The capacity $c = 384$ provides sufficient security against generic attacks for both 128- and 256-bit keys. As explained in Section 2, we know from [10] that the generic security level is 2^{128} for a 128-bit key and 2^{256} for a 256-bit key.

4.2 Permutation f

Our underlying sponge function f is a permutation and thus is invertible. For compactness, we specify only the forward permutation here as the inverse is not required for practical purposes.

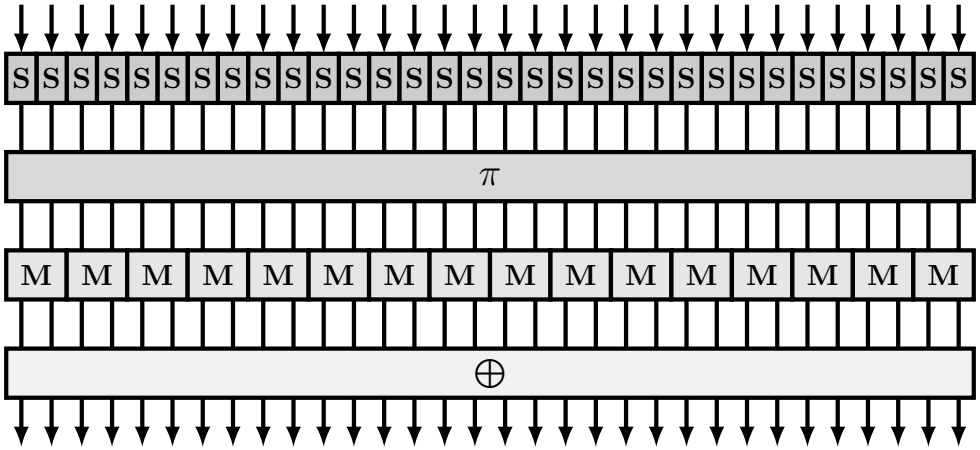

Figure 3: A single round of sponge permutation f

The permutation consists of a number of rounds. As shown in Figure 3, each round consists of four steps: a substitution, a bitwise permutation, a mixing layer, and the addition of a round constant.

4.2.1 Substitution Step

The substitution step is a bricklayer permutation that uses 32 identical, bijective 16×16 S-boxes. This step is the main source of confusion within the permutation. Furthermore, it is the only non-linear step, as is typical with many substitution-based symmetric key algorithms [1].

To the best of our knowledge this is the first cryptosytem to use such large S-boxes. We believe that, at the time of writing, the largest S-boxes used in the literature are the 8×8 bijective S-boxes used by the Advanced Encryption Standard (AES) [12][13].

Our S-box is an AES-inspired design taken directly from Wood's thesis on the subject [4]. The primary reason for using this particular class of 16-bit S-boxes is that they are efficiently implementable in hardware. Rather than being based on a random mapping, they are based on multiplicative inversion in a finite field followed by an affine transformation. This allows us to implement a circuit which performs the field operations rather than use the corresponding (and prohibitively large) look-up table.

This S-box is based on multiplicative inversion in $\mathrm{GF}(2^{16})/\langle p(x)\rangle$ where

$$p(x) = x^{16} + x^5 + x^3 + x + 1.$$

We represent an input to the S-box (and inverse S-box) as a 16-bit column vector

$$x = \begin{pmatrix} x_{15} & x_{14} & \dots & x_1 & x_0 \end{pmatrix}^{\mathrm{T}},$$

where x_{15} is the MSB. Using this notation, the forward S-box function is given as

$$\mathbf{S}(x) = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_{15} \\ x_{14} \\ x_{13} \\ x_{12} \\ x_{11} \\ x_{10} \\ x_9 \\ x_8 \\ x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{pmatrix}^{-1} \oplus \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

This S-box was chosen among others discovered in Wood's thesis because it has the smallest hardware footprint. A hardware implementation for this S-box requires just 1238 XOR gates and 144 AND gates [4].

4.2.2 Bitwise Permutation Step

Bitwise permutations are easily implementable in hardware via a simple rerouting of wires. Compared to a permutation on the words of the state, a bitwise permutation intuitively provides much better diffusion. The bitwise permutation step is the main source of long-range (i.e. across the entire state) diffusion in the algorithm.

The bitwise permutation also helps maximize the minimum number of active S-boxes by being subject to certain constraints. We use a permutation that satisfies the following properties:

1. All outputs of a given S-box go to 16 different mixers
2. The permutation is a *derangement*; it has no fixed points
3. High order; it does not repeat within the number of rounds
4. No low order bits; the order of any bit equals the order of the overall permutation
5. Definable by an affine function

The *order* of a bitwise permutation is the number of times it must be applied before it ends up in its original orientation. A particular bit in a bitwise permutation also has the notion of order and a bit of low order is one that returns to its original position before the entire permutation begins to cycle. If a bit has order zero, it is unaffected by the bitwise permutation and is called a *fixed point*.

We chose the following permutation to use for our algorithm since it satisfies all properties:

$$\pi(x) = 31x + 15 \pmod{512}$$

This bitwise permutation has order 32. For a complete listing of such bitwise permutations that satisfy our requirements, refer to [14].

4.2.3 Mix Step

The purpose of the mix step is to provide local diffusion (i.e. across two words) and increase the linear and differential branch numbers of a round from two to three. We use a mixer based on multiplication by a 2×2 matrix in $\mathrm{GF}(2^{16})$ modulo the irreducible polynomial

$$p(x) = x^{16} + x^5 + x^3 + x^2 + 1.$$

The mixer takes two words A and B as input and produces outputs A' and B' as follows:

$$\begin{pmatrix} A' \\ B' \end{pmatrix} = \begin{pmatrix} 1 & x \\ x & x+1 \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix}$$

The MSB of each word is taken as the leftmost bit and is represented by x^{15}.

The forward mixer is efficiently implementable in hardware. Figure 4 shows how this matrix multiplication is implemented. The $x*$ operation is a multiplication by x in $\mathrm{GF}(2^{16})$.

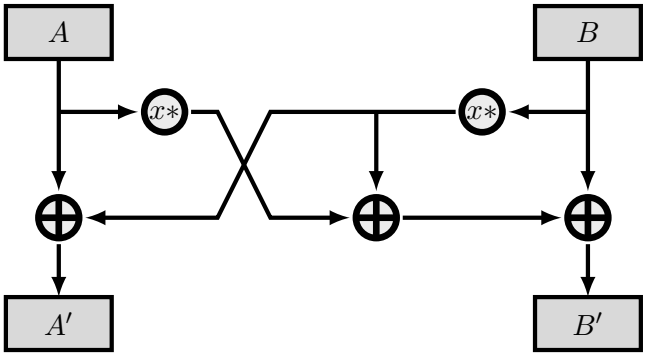

Figure 4: Hardware implementation of the forward mixer function.

4.2.4 Add Round Constant Step

This step consists of adding a 512-bit value to the state using bitwise XOR in order to disrupt symmetry and prevent slide attacks. The round constant RC_i for round i is given by the formula

$$RC_i = \mathbf{SHA3\text{-}512}(\mathbf{ASCII}(i)),$$

where $\mathbf{ASCII}(i)$ is a function that provides the one or two byte ASCII representation of i and $\mathbf{SHA3\text{-}512}$ is the SHA-3 hash function that outputs a 512-bit message digest.

4.3 Number of Rounds

This algorithm uses 10 rounds for a 128-bit key and 16 rounds for a 256-bit key. The number of rounds is determined, as is typical with block ciphers and permutations, by calculating the number needed for resistance to linear and differential cryptanalysis and adding some buffer to increase the security margin. For a more in-depth treatment, refer to Section 5.

4.4 Customization

While a specific instantiation is given here, our algorithm is highly customizable within our security margin. This could be useful in the case that different users want unique, proprietary algorithms. We list several possible customizations here.

4.4.1 State Initialization

Modifying the initial value of the inner state is the primary and easiest method of customization. In the given specification, the inner state (like the outer state) is initialized to zero. This is not a requirement; indeed, the inner state could be initialized to any 384-bit value. Each user could generate their own unique value to set during the initialization phase. This happens before the first mute calls that absorb the key.

4.4.2 S-boxes

The AES-inspired S-box used here is efficiently implementable in hardware. There are certainly many other cryptographically secure 16-bit S-boxes, but randomly generated ones may not be suitable for hardware implementation due to size constraints. This is an area for further research. Still, several other AES-like 16-bit S-boxes are presented in [4]. Any new S-box introduced into the algorithm shall be analyzed to determine its linear and differential characteristics and the number of rounds should be adjusted accordingly if necessary. This analysis can easily be performed using the tools mentioned in Section 5.

4.4.3 Bitwise Permutations

The bitwise permutation provided in this algorithm specification is one of many that satisfies the constraints we impose. For a complete listing of all suitable bitwise permutations we refer to [14]. Users may select any of these without need for further cryptanalysis.

4.4.4 Mixers

Our mixer is based on a specific 2×2 matrix multiplication in $\mathrm{GF}(2^{16})$ modulo a specific irreducible polynomial $p(x)$. Many matrices are expected to satisfy the constraints that we impose. These constraints are:

1. The 2×2 matrix should be invertible in $\mathrm{GF}(2^{16}) / \langle p(x) \rangle$
2. The matrix should have differential and linear branch number equal to three (the maximum possible)

Like the addition of a new S-box, any new matrix introduced to the algorithm should be analyzed to ensure it meets these constraints. This analysis can also easily be performed using the tools mentioned in Section 5.

4.4.5 Round Constants

The round constants presented here are based on SHA-3 hash values. However, they could be any values that satisfy the following constraints. Round constants should be:

1. Unique for each round; to prevent against slide attacks
2. Random, pseudorandom, or highly asymmetric; to reduce symmetry in the state

The round constants are not expected to have any cryptographic significance outside of this. Different users can generate their own unique set of round constants without difficulty.

5 Comments on Cryptanalysis

The duplex construction has been shown to be secure against generic attacks by the KECCAK team [7]. Therefore it is sufficient for us to assess only the security of the underlying sponge permutation f.

We provide an overview of our preliminary cryptanalysis here. Emphasis is placed on the most general and prevalent forms of attacks. Resistance against these techniques should result in resistance against many other less general techniques. Our aim is to provide intuitive explanations of prevalent methods and simply explain why our permutation should be resistant. Further cryptanalysis, as with all cryptosystems, is always welcome for future work.

5.1 Differential Cryptanalysis

Differential cryptanalysis was publicly introduced by Biham and Shamir in 1991 in their landmark paper on the subject [15]. Since then, it has been applied with varying degrees of success to a great number of cryptosystems. As such, it is a fundamental requirement of symmetric key cryptosystem design to prove resistance to differential cryptanalysis.

To determine the resistance of our algorithm to differential cryptanalysis, we first have to determine the maximum differential probability of our S-box. We determined this value to be $p_{D,max} = 2^{-14}$ using an S-box evaluation program called `Eval16BitSbox` [16] written with Kaminsky's Parallel Java 2 library [17].

Next, it is necessary to determine the differential branch number of a round. For this, we only need to analyze the mixer. We purposefully designed a mixer with differential branch number equal to three, meaning that minimally three S-boxes will be differentially active between two rounds. This is in fact the maximum achievable branch number for a transformation defined by multiplication by a 2×2 matrix. To verify this, we used a SAT solver called CryptoMiniSat [18]. This SAT solver takes as input a Boolean equation in conjunctive normal form (CNF) and determines if it is *satisfiable*; that is, if it can ever produce an output of '1' for any set of input values. Our CNFs were generated using Kaminsky's `SatProblem` Java class [16].

The CNFs generated are unsatisfiable if and only if the mixer has differential branch number equal to three since it answers the following question: *is it possible to have a difference in only one input and only one output?* Through SAT solver analysis we determined that this is not possible for our mixer; that is, if there is a non-zero difference in only one input, there must be a non-zero difference in each output. In the event that there is a difference in both inputs, there may be a

difference in only one output. This still leads to a differential branch number of three since two S-boxes must have been active in the previous round to lead to those two input differences. The probability of a difference in either output is $p_{D,out} = 2^{-15}$.

With all of this information, it is possible to calculate the number of rounds needed for resistance to differential attacks. The worst-case probability of successfully propagating a difference over two rounds is given by

$$(p_{D,max})^{\mathcal{B}_D} \cdot p_{D,out},$$

where $\mathcal{B}_D = 3$ is the differential branch number. Extending on this, we found that the complexity of a differential attack exceeds the complexity of a brute force attack at six rounds for a 128-bit key. To increase our security margin significantly, we require 10 rounds for a 128-bit key. For a 256-bit key, 16 rounds are required to achieve a similar security margin.

5.2 Linear Cryptanalysis

Linear Cryptanalysis was first introduced by Matsui in 1993 in his landmark paper, [19]. As with differential cryptanalysis, it is a fundamental requirement of symmetric key cryptosystem design to prove resistance against linear attacks.

Recall that we have verified via SAT solver analysis that the differential branch number of our mixer is three. In [12], Daemen and Rijmen prove the following result: the linear branch number of a linear transformation specified by multiplication by a matrix M is equal to the differential branch number of the linear transformation specified by the transpose of that matrix. Therefore, a sufficient condition for the differential and linear branch numbers to be equal is that the matrix is symmetric. Our matrix is symmetric, and therefore we know $\mathcal{B}_D = \mathcal{B}_L$ without the need for further analysis.

The final step to prove the resistance of our algorithm against linear cryptanalysis involves determining the linear bias of two complete rounds of our permutation. To combine linear biases, we use Matsui's Piling-Up Lemma from [19]:

$$\epsilon = 2^{n-1} \prod_{i=1}^{n} \epsilon_i,$$

where $n = 3$ is the number of linearly active S-boxes across two rounds and $\epsilon_i = \epsilon_{L,max} = 2^{-8}$ is the worst case linear bias of those S-boxes. Also from Matsui's paper, we know that the number of plaintext/ciphertext pairs (again referred to loosely as the *complexity*) needed to exploit the overall bias ϵ is approximately ϵ^{-2}. Using this information, we determined that the complexity of a linear attack exceeds the complexity of a brute force search of a 128-bit keyspace at six rounds. To increase our security margin significantly, we require 10 rounds for a 128-bit key. For a 256-bit key, 16 rounds are required to achieve a similar security margin.

5.3 Algebraic Attacks

Differential and linear cryptanalysis take a probabilistic approach to estimating the behavior of a system. In contrast, algebraic attacks take a deterministic approach in that they aim to find mathematical models of a system that hold with unity probability. For example, in 2001 Ferguson et al. [20] introduced an elegant and complete algebraic representation of AES. The ability to create such a simple mathematical representation of the cipher initially raised alarm throughout the cryptographic community. However, the security of AES seems to be uncompromised since we believe it

is far too difficult to solve such an algebraic system - the algebraic complexity of the AES S-box is too high.

Until there is reason to believe otherwise, it seems that these algebraic attacks would be highly ineffective against our permutation. Even if there were a practical algebraic attack demonstrated on AES, which all literature indicates as highly implausible right now, the much larger size of our S-box and therefore the much higher algebraic complexity (see [4]) leads us to conjecture that our permutation would still be resistant.

6 Conclusions

In this paper we presented a customizable authenticated encryption algorithm based on the duplex construction that is targeted for hardware implementation. We believe this algorithm to be highly secure against known attacks. In particular, we provided proof of resistance against linear and differential attacks as well as solid reasoning for resistance against algebraic attacks.

6.1 Future Work

There are two primary areas for further work relating to the algorithm presented here. As with all cryptosystems, further cryptanalysis is always appreciated. In particular, it would be interesting to determine the best possible linear and differential trails across several rounds.

The other area of work is the hardware implementation of the algorithm described here. In particular, quantitative results relating to the resource usage for an FPGA implementation are of great interest. We are confident that our permutation is designed in such a way that a relatively small amount of resources will be required.

References

[1] D. Stinson, *Cryptography: Theory and Practice, Second Edition*. CRC/C&H, 3rd ed., 2006.

[2] N. Ferguson, B. Schneier, and T. Kohno, *Cryptography Engineering*. Indianapolis, IN: Wiley Publishing, 2010.

[3] M. T. Kurdziel and J. Fitton, "Baseline Requirements for Government and Military Encryption Algorithms," in *MILCOM 2002. Proceedings*, vol. 2, pp. 1491–1497, IEEE, 2002.

[4] C. A. Wood, "Large Substitution Boxes with Efficient Combinational Implementations," Master's thesis, Rochester Institute of Technology, 2013. `http://scholarworks.rit.edu/theses/5527/`.

[5] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "The KECCAK reference," *NIST SHA-3 Submission Document*, January 2011. `http://http://keccak.noekeon.org/Keccak-reference-3.0.pdf`.

[6] S.-j. Chang, R. Perlner, W. E. Burr, M. S. Turan, J. M. Kelsey, S. Paul, and L. E. Bassham, "Third-Round Report of the SHA-3 Cryptographic Hash Algorithm Competition." NIST Internal Report 7896, November 2012. `http://nvlpubs.nist.gov/nistpubs/ir/2012/NIST.IR.7896.pdf`.

[7] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Cryptographic Sponge Functions," 2011. http://http://sponge.noekeon.org/CSF-0.1.pdf.

[8] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Duplexing the sponge: single-pass authenticated encryption and other applications," in *Selected Areas in Cryptography*, pp. 320–337, Springer, 2012.

[9] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Duplexing the sponge: single-pass authenticated encryption and other applications," August 2010. http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/presentations/DAEMEN_SpongeDuplexSantaBarbaraSlides.pdf.

[10] P. Jovanovic, A. Luykx, and B. Mennink, "Beyond $2^{c/2}$ Security in Sponge-Based Authenticated Encryption Modes." Cryptology ePrint Archive, Report 2014/373, 2014. http://eprint.iacr.org/.

[11] E. Barker and A. Roginsky, "Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths." NIST Special Publication 800-131A, January 2011. http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf.

[12] J. Daemen and V. Rijmen, *The Design of Rijndael: AES-The Advanced Encryption Standard*. Springer, 2002.

[13] NIST, "Specification for the Advanced Encryption Standard (AES)." Federal Information Processing Standards Publication 197, November 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[14] M. Kelly, "Design and Cryptanalysis of a Customizable Authenticated Encryption Algorithm," Master's thesis, Rochester Institute of Technology, 2014. http://scholarworks.rit.edu/theses/8325/.

[15] E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," *Journal of CRYPTOLOGY*, vol. 4, no. 1, pp. 3–72, 1991.

[16] A. Kaminsky, "Block Cipher Analysis." http://cs.rit.edu/~ark/parallelcrypto/blockcipheranalysis/, 2014.

[17] A. Kaminsky, "Parallel Java 2 Library." http://cs.rit.edu/~ark/pj2.shtml, 2014.

[18] M. Soos, "CryptoMiniSat." http://msoos.org/cryptominisat2/, 2014.

[19] M. Matsui, "Linear Cryptanalysis Method for DES Cipher," in *Advances in Cryptology—EUROCRYPT'93*, pp. 386–397, Springer, 1994.

[20] N. Ferguson, R. Schroeppel, and D. Whiting, "A Simple Algebraic Representation of Rijndael," in *Selected Areas in Cryptography*, pp. 103–111, Springer, 2001.