The CS^2 Block Cipher*

Tom St Denis

Secure Science Corporation
tom@securescience.net

Abstract. In this paper we describe our new CS^2 block cipher which is an extension of the original CS-Cipher. Our new design inherits the efficiency of the original design while being upgraded to support a larger block size, key size as well as use a slightly improved substitution box. We prove that our design is immune to differential and linear cryptanalysis as well as argue it resists several other known attacks.

Keywords. Secret Key-Cryptography, Pseudo-Hadamard Transform.

1 Introduction

In [2] the CS–Cipher was introduced. It is a 64–bit block cipher designed for efficiency on 8–bit processors as well as in hardware. In their design they introduced the use of multipermutations [4] as a design construct as well as a very non-linear round transform. In a subsequent paper [3] Vaudenay claimed that the active substitution box "sbox" count is at least 72 over the full cipher, thus concluding the design is immune to differential and linear cryptanalysis. As of yet no full cryptanalysis of the CS–Cipher is known to exist.

In this paper we extend the CS–Cipher using the results of [5] and [7] to produce an efficient 128–bit block cipher with a 128–bit secret key. We show how our design can be implemented efficiently as well as explain the hardware rationale. We also prove that our design is immune to both differential and linear cryptanalysis. We further argue that our design resists several other known attacks.

2 Design

The CS^2 block cipher is a substitution–permutation network (SPN) which accepts a 128–bit plaintext P and produces a 128–bit ciphertext C. It accepts a 128–bit key K which is processed by our key schedule to produce the thirty–three round keys $K_{0..32}$ required.

* Moniker used with permission from Serge Vaudenay.

2.1 Non-Linear Substitution γ

The non-linear substitution γ is by far the most critical part of our design because its size and delay greatly affect the cipher as a whole. We had to strike a balance between efficiency and security to ensure the overall design was both efficient, practical and secure. Ultimately we decided to use a 4×4 sbox $\hat{\gamma}$ in a SPN like construction similar to the designs of Twofish and WHIRLPOOL.

x	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
$\hat{\gamma}(x)$	8_x	4_x	0_x	D_x	A_x	7_x	6_x	2_x	B_x	5_x	3_x	9_x	1_x	F_x	C_x	E_x

Fig. 1. Non-Linear Substitution $\hat{\gamma}(x)$

Let H_1 represent the FPHT over $GF(2)[x]/(x^4 + x^3 + 1)$.

$$
\begin{array}{|c|}
\hline
a_0, b_0 = \lfloor x/16 \rfloor,\ x \bmod 16 \\
a_1, b_1 = \hat{\gamma}(a_0),\ \hat{\gamma}(b_0) \\
a_2, b_2 = H_1 \circ \langle a_1, b_1 \rangle \\
a_3, b_3 = \hat{\gamma}(a_2),\ \hat{\gamma}(b_2) \\
y = 16a_3 + b_3 \\
\hline
\end{array}
$$

Fig. 2. Non-Linear Substitution $y = \gamma(x)$

2.2 Non-Linear Multipermutation $\theta(x)$

As in the CS-Cipher we have used a $(2, 3)-$multipermutation with two differences. We applied the non-linear substitution before the linear transform and we use the H_1 transform over $GF(2)[x]/(x^8 + x^4 + x^3 + x + 1)$ instead of a linear transform over $\mathbb{Z}_2^{16}$. The overall concept and purpose remains the same.

$$
\begin{array}{|c|}
\hline
a_0, b_0 = \lfloor x/256 \rfloor,\ x \bmod 256 \\
a_1, b_1 = \gamma(a_0),\ \gamma(b_0) \\
a_2, b_2 = H_1 \circ \langle a_1, b_1 \rangle \\
y = 256a_2 + b_2 \\
\hline
\end{array}
$$

Fig. 3. Non-Linear Multipermutation $y = \theta(x)$.

2.3 Round Transform $\zeta(P, r, K)$

The round transform ζ is the H_4 transform implemented in a network mode (fig. 4). The H_1 transforms have been replaced by the non-linear multipermutation θ. Before layer l of the network the round key K_{4r+l} is exclusively or'ed against the block. To describe the transform we shall make use of the coordinate pairing tables $\rho_{0..3}$ (fig. 5). Let $\rho_{p,q,r}$ represent the p'th ρ table, q'th pair and the r'th value of the pair. For example, $\rho_{3,7,0} = 7$.

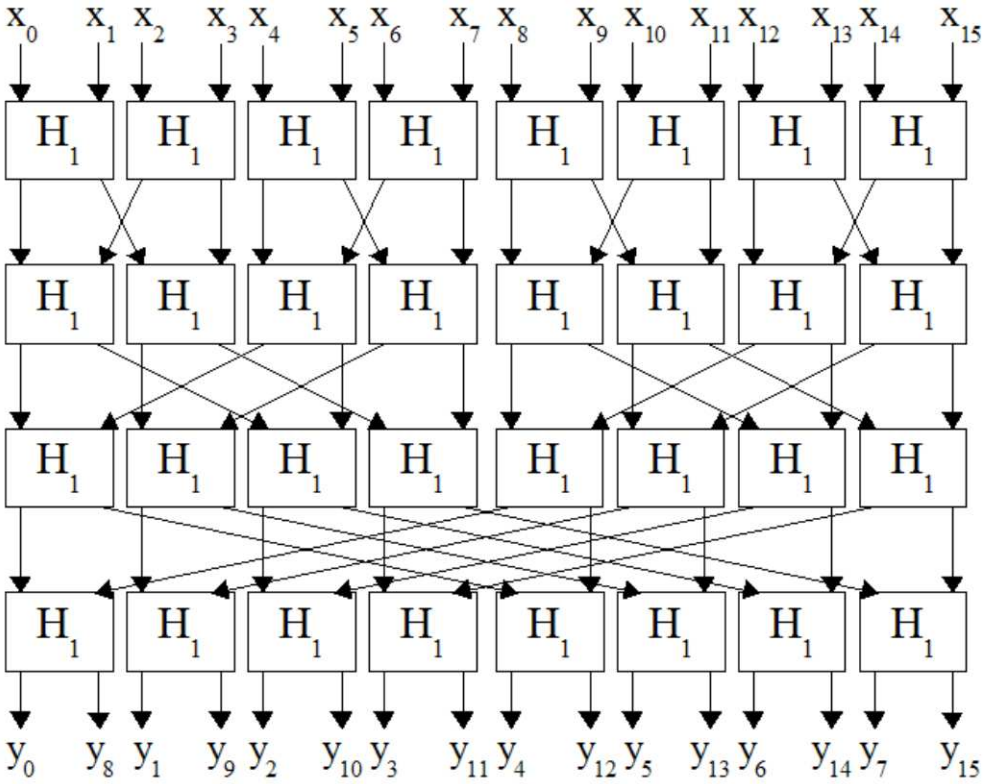

Fig. 4. The Round Transform Network

ρ_0	$0,1$	$2,3$	$4,5$	$6,7$	$8,9$	$10,11$	$12,13$	$14,15$
ρ_1	$0,2$	$1,3$	$4,6$	$5,7$	$8,10$	$9,11$	$12,14$	$13,15$
ρ_2	$0,4$	$1,5$	$2,6$	$3,7$	$8,12$	$9,13$	$10,14$	$11,15$
ρ_3	$0,8$	$1,9$	$2,10$	$3,11$	$4,12$	$5,13$	$6,14$	$7,15$

Fig. 5. Coordinate Pairings Tables ρ

1. for $k = 0$ to 3 do
   1.1 $x \leftarrow x \oplus K_{4r+k}$
   1.2 for $m = 0$ to 7 do
      1.2.1 $y \leftarrow \theta(256x_{\rho_{k,m,0}} + x_{\rho_{k,m,1}})$
      1.2.2 $x_{\rho_{k,m,0}} \leftarrow \lfloor y/256 \rfloor, x_{\rho_{k,m,1}} \leftarrow y \bmod 256$
2. return x.

Fig. 6. Round function $y = \zeta(x, r, K)$

2.4 Key Schedule

The key schedule (fig.7) is responsible for accepting a 128–bit input key and producing the thirty–three required round keys. Let K_A represent the secret key and $K_{A,j}$ represent the j'th octet of the secret key. Let σ_i represent the i'th 128–bit constant and $\sigma_{i,j}$ represent the j'th octet of the i'th constant defined by (1).

$$\sigma_{i,j} = \gamma(\gamma(i) \oplus j), 0 \leq i < 32, 0 \leq j < 16 \tag{1}$$

In this design the we use consecutive values of σ as the layer keys for the round function ζ in order to produce the round keys. For the purposes of efficiency we create a modified version of ζ called $\hat{\zeta}$ which emits the intermediate outputs of every layer as round keys.

1. for $k = 0$ to 7 do
   1.1 $\{K_{4k+0}, K_{4k+1}, K_{4k+2}, K_{4k+3}\} \leftarrow \hat{\zeta}(K_A, \sigma_{4k})$
   1.2 $K_A \leftarrow K_{4k+3}$
2. $K_{32,j} \leftarrow \gamma(K_{A,j}), 0 \leq j \leq 15$

Fig. 7. Key Scheduler

2.5 Block Cipher

The entire cipher is simply eight rounds of the ζ round function followed by a final post-whitening key step.

1. for $k = 0$ to 7 do
   1.1 $x \leftarrow \zeta(x, k, K)$
2. return $x \oplus K_{32}$.

Fig. 8. CS^2 Block Cipher

We note that no decryption routine is described. While the cipher is a bijection a decryption routine was not deemed necessary since many useful modes of operations such as CTR, OMAC and EAX only make use of the forward direction of the cipher.

3 Design Rational

This cipher has been designed primarily with hardware and embedded software in mind. Most of the design choices have been related to the gate depth of the round function as well as the code size on the typical 8-bit processor (such as the Intel 8051). There are a variety of implementation options as well.

3.1 Non-Linear Substitution γ

The non-linear substitution γ was designed to have an efficient hardware implementation while still having ideal cryptographic properties. The function $\hat{\gamma}$ can be written as the following multivariate equations.

$$\begin{aligned} y_0 &= x_0x_1 \oplus x_0x_2 \oplus x_3 \oplus x_0x_1x_3 \oplus x_0x_2x_3 \oplus x_1x_2x_3 \\ y_1 &= x_2 \oplus x_3 \oplus x_0x_3 \\ y_2 &= x_0 \oplus x_1x_2 \oplus x_0x_1x_3 \\ y_3 &= 1 \oplus x_0 \oplus x_1 \oplus x_0x_1x_2 \oplus x_2x_3 \end{aligned} \tag{2}$$

The H_1 function can be written easily as well.

$$\begin{aligned} y_0 &= x_3 \oplus x_4 \\ y_1 &= x_0 \oplus x_5 \\ y_2 &= x_1 \oplus x_6 \\ y_3 &= x_2 \oplus x_7 \oplus x_3 \\ y_4 &= x_4 \oplus x_0 \\ y_5 &= x_5 \oplus x_1 \\ y_6 &= x_6 \oplus x_2 \\ y_7 &= x_7 \oplus x_3 \end{aligned} \tag{3}$$

The function was also designed to have the following cryptographic properties.

1. An $DP_{max}(\gamma)$ of $\frac{10}{256} \approx 2^{-4.67}$
2. An $LP_{max}(\gamma)$ of $\frac{16}{256} = 2^{-4}$
3. An algebraic degree of six.
4. No fixed ($\gamma(x) = x$) or points of involution ($\gamma(\gamma(x)) = x$).

The low differential and linear profiles help prevent high probability trails (and hulls) from forming. Even though the design primarily relies on the wide–trail design philosophy it was still thought important to minimize such characteristics. The algebraic degree has been raised to the maximum value of six for such a function to help prevent trivial algebraic style attacks. Even though no such attacks against CS–Cipher are known it was another design consideration. Finally fixed points and points of involution were avoided to prevent fixed points in the round transform. They also guarantee that K_{31} and K_{32} are distinct.

In hardware the most practical design implementation would be to compute γ as a function of $\hat{\gamma}$ and H_1. This will result in a very low gate count compared to what an 8×8 ROM would require and have a reasonable gate delay. In software γ is best implemented as an array placed in ROM.

	CS^2-Cipher	**CS-Cipher**
DP_{max}	$2^{-4.67}$	2^{-4}
LP_{max}	2^{-4}	2^{-4}
Algebraic Degree	6	3
Nonlinear Order	3	3

Fig. 9. Comparison between substitution boxes of CS^2-Cipher and CS-Cipher.

3.2 Non-Linear multipermutation θ

The non-linear multipermutation was also designed to be efficient in hardware to implement. Assuming that the two γ substitutions are implemented in parallel the overall gate depth is not much higher than that of the substitutions themselves. The following equations give the H_1 function as it applies to this transform.

$$
\begin{aligned}
y_0 &= x_8 \oplus x_7 \\
y_1 &= x_0 \oplus x_9 \oplus x_7 \\
y_2 &= x_1 \oplus x_{10} \\
y_3 &= x_2 \oplus x_{11} \oplus x_7 \\
y_4 &= x_3 \oplus x_{12} \oplus x_7 \\
y_5 &= x_4 \oplus x_{13} \\
y_6 &= x_5 \oplus x_{14} \\
y_7 &= x_6 \oplus x_{15} \\
y_8 &= x_8 \oplus x_0 \\
y_9 &= x_9 \oplus x_1 \\
y_{10} &= x_{10} \oplus x_2 \\
y_{11} &= x_{11} \oplus x_3 \\
y_{12} &= x_{12} \oplus x_4 \\
y_{13} &= x_{13} \oplus x_5 \\
y_{14} &= x_{14} \oplus x_6 \\
y_{15} &= x_{15} \oplus x_7
\end{aligned}
\tag{4}
$$

3.3 Round Function ζ

The round function ζ was chosen for its efficient implementation which yields a gate delay of four times the gate delay of the θ function. As a result the round function has a relatively low gate delay. The coordinate pairings table was designed to create a Fast pseudo-Hadamard Transform (FPHT) [5] so that it has a provable branch and two round trail weight.

Our choice of the FPHT was motivated by the fact that like the SHARK [6] design it resists the saturation and truncated differential attacks very well. That is, since every output coordinate is a non-null function of all of the input coordinate diffusion is optimal. However, unlike the SHARK design our round transform has a more efficient $O(\log n)$ time implementation and nearly twice as many active coordinates over two rounds.

In hardware the best approach in terms of space and time complexity would be to implement ζ as a pipeline. The transform network has four layers which could accommodate up to four blocks at a time (suitable for CTR mode) with only a modest addition to the design. In each layer the eight θ transforms could be implemented in parallel since they are very small.

In software the ideal approach in terms of space would be to implement the four layers of the transform network as separate subroutines that could be called both by $\hat{\zeta}$ and ζ. In terms of speed it would be ideal to inline the layers to save on index pointer setup and stack operation time.

3.4 Key Schedule

The key schedule has also been designed with both platforms in mind. The original key schedule designs allowed up to a 256–bit key, had round constants

based off of the sine function and used a full call to ζ to produce a single 128–bit round key word. Not only did this severely penalize on–the–fly round key computation but it also made the design larger and hindered cryptanalysis.

Our original 256–bit key designs were designed to use the round function to lower the design, implementation and cryptanalysis time. However, all of our attempts were either weak against reduced round related keys attacks or were too inefficient for on–the-fly computation. As a result for this design we reduced the key size to 128–bits.

The round constants can both be computed easily on–the–fly from γ and also stored in ROM for quick access. The nested nature (eqn. 1) of the round keys allows a minimal 17 table lookups by first computing $\gamma(i)$ and using it for the other octets of the i'th round key constant.

The round keys are extracted after every layer of $\hat{\zeta}$ so that they can computed on–the–fly with just over the cost of one encryption (with pre–computed round keys). In hardware, the round keys can be computed within each pipeline stage just before they are required. That is, each pipeline stage would have two buffers. One buffer stores the round key to be used in that stage against the plaintext and the other holds the computed (at the same time as the plaintext is encrypted) round key for the next round. At the end of each pipeline stage the plaintext and new round key are forwarded to the next pipeline stage. As a result, with over twice the gate count and a single pipeline cycle penalty per plaintext the circuit can encrypt up to four plaintexts under up to four different keys at once.

4 Cryptanalysis

4.1 Linear and Differential Cryptanalysis

In [7] it was proved that two rounds of ζ will have at least twenty-four active coordinates. As a result both linear and differential cryptanalysis become ineffective quickly. With six rounds any four round trail will have a probability far too low to be useful. Eight rounds provides a large margin of security from both attacks.

Rounds in Attack	**Differential Cryptanalysis**	**Linear Cryptanalysis**
2	2^{112}	2^{73}
4	2^{224}	2^{145}
6	2^{336}	2^{216}

Fig. 10. Plaintext Requirements for Linear and Differential Cryptanalysis.

4.2 Truncated Differentials

In truncated differential attacks the goal is to predict only part of the output difference or more importantly predict a pattern of difference propagation. For example, the pattern $\Delta_x \rightarrow \Delta_y : \langle a, b\rangle \rightarrow \langle 0, c\rangle$ for non–zero a, b, c is a valid truncated differential for the θ function. In the attack a truncated differential that does not have uniform probability is exploited to create a key distinguishing attack on the last round of the cipher.

For example, in the case of the two point pseudo-Hadamard Transform over the integers modulo 256 the input difference $\langle 128 + a, 0\rangle$ causes the output difference $\langle 0, b\rangle$ for $0 < a < 128$ and any non–zero b with a probability of one. For a truly random permutation you would expect this to occur with a probability of $\frac{1}{256}$ and such a bias can be exploited.

In the case of CS^2 we used a $(2, 3)-$multipermutation which achieves close to ideal properties. The difference $\langle a, b\rangle$ for non–zero a, b shall behave as expected for a random function. It shall emit differences of $\langle 0, c\rangle$ and $\langle d, 0\rangle$ for the appropriate $2\sqrt{\#\theta} - 2$ as would be expected from a random function. The only flaw is that an input difference with one zero shall always emit an output with two non–zero coordinates. As of yet no known exploit of this property is known to exist.

4.3 Higher Order Differential Cryptanalysis

The nonlinear order of the γ function is three which means there exists third order derivatives which produce a constant output. It is thought that this attack cannot defeat more than three rounds of the design due to the fact that the γ function has been placed within the ζ round transform not just at the edge.

4.4 Slide and Boomerang Attacks

The slide attack is not applicable as every round and every layer have their own round keys. Also during the key scheduling every emitted key is based on a unique grouping of σ constants. The constants also help prevent fixed points in the key schedule from emitting short periodic round keys.

While the decryption routine is not defined in this paper it can be shown to have the same two round trail weight as the forward direction. Therefore, it does not seem likely that the boomerang attack applies to this design.

4.5 Related Key Attacks

In a related key attack the attacker attempts to learn information about the key through known or chosen plaintext and a relationship between two keys. It is unlikely that such an attack would prove useful since the key schedule uses the non-linear mixing network $\hat{\zeta}$. Any two rounds are guaranteed to have at least 24 active key coordinates. Therefore, it is unlikely that a related key attack combined with known or chosen plaintext could work effectively.

5 Conclusion

A simple, efficient and secure block cipher has been proposed. It was designed after the CS block cipher as well as the research into FPHT transforms of [5] and [7]. We feel that the design is a reasonable alternative to Rijndael for hardware platforms since it is equally as efficient and does not rely on a highly algebraic non-linear transform.

References

[1] J. Daemen and L. Knudsen and V. Rijmen, "The Block Cipher SQUARE", Fast Software Encryption, v.1267 of Lecture Notes in Computer Science, pp. 149–165. Springer-Verlag, 1997

[2] J. Stern and S. Vaudenay. CS-Cipher. In Fifth International Workshop on Fast Software Encryption, Berlin, Germany, March 1998. Springer-Verlag.

[3] S.Vaudenay, "On the Security of the CS-Cipher", Fast Software Encryption, March 1999, Springer-Verlag, pp. 260-274

[4] S. Vaudenay, "On the Need for Multipermutations: Cryptanalysis of MD4 and SAFER", LIENS - 94 - 23, November 1994

[5] T. St Denis, "Fast Pseudo-Hadamard Transforms", Cryptology ePrint Archive, Report 2004-010

[6] A. Bossalaers, J. Daemen, B. Preneel, V. Rijmen, E. De Win, "The Cipher SHARK", Fast Software Encryption, pp. 99–111, 1996.

[7] T. St Denis, "The CSQUARE Transform", Cryptology ePrint Archive, Report 2004-026

Appendix A - Test Vectors

```
KEY: 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
PT : 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
CT : 58 70 f4 0b d1 57 00 ff 1e 66 bb b7 0b c3 6e 50

KEY: 0f 0e 0d 0c 0b 0a 09 08 07 06 05 04 03 02 01 00
PT : 0f 0e 0d 0c 0b 0a 09 08 07 06 05 04 03 02 01 00
CT : ca f2 58 81 d8 da fa 5f 16 a1 8a 0e f5 92 e3 f8
```

This article was processed using the LaTeX macro package with LLNCS style