Dismantling the AUT64 Automotive Cipher

Christopher Hicks, Flavio D. Garcia and David Oswald

School of Computer Science
University of Birmingham
United Kingdom
{c.hicks,f.garcia,d.f.oswald}@cs.bham.ac.uk

Abstract. AUT64 is a 64-bit automotive block cipher with a 120-bit secret key used in a number of security sensitive applications such as vehicle immobilisation and remote keyless entry systems. In this paper, we present for the first time full details of AUT64 including a complete specification and analysis of the block cipher, the associated authentication protocol, and its implementation in a widely-used vehicle immobiliser system that we have reverse engineered. Secondly, we reveal a number of cryptographic weaknesses in the block cipher design. Finally, we study the concrete use of AUT64 in a real immobiliser system, and pinpoint severe weaknesses in the key diversification scheme employed by the vehicle manufacturer. We present two key-recovery attacks based on the cryptographic weaknesses that, combined with the implementation flaws, break both the 8 and 24 round configurations of AUT64. Our attack on eight rounds requires only 512 plaintext-ciphertext pairs and, in the worst case, just $2^{37.3}$ offline encryptions. In most cases, the attack can be executed within milliseconds on a standard laptop. Our attack on 24 rounds requires 2 plaintext-ciphertext pairs and $2^{48.3}$ encryptions to recover the 120-bit secret key in the worst case. We have strong indications that a large part of the key is kept constant across vehicles, which would enable an attack using a single communication with the transponder and negligible offline computation.

Keywords: Automotive security · Hardware and software reverse engineering

1 Introduction

Since 1995, it has been mandatory for vehicle manufacturers who wish to sell their vehicles inside the EU to fit them with an electronic immobiliser [Com95]. It has been estimated that between 1995 and 2008, vehicle immobilisers have reduced the rate of vehicle theft by 40% [OV16].

Vehicle immobilisers are electronic devices which prevent the engine of a vehicle from starting when the corresponding transponder is not present. A transponder is a wireless authentication device, which is usually embedded into the plastic of a vehicle key. The immobiliser authenticates the transponder using an antenna mounted around the ignition barrel so that the immobiliser can communicate with the transpon-

Make	Models	Years
Mazda	323	1999-2003
	626	1999-2002
	Demio	1999-2001
	Miata/MX-5	2000-2005
	MPV	2000-2006
	Premacy	2000-2004
	121	1999-2001
	BT-50	2006
Ford	Ranger	2006
Proton	415	1998
	416	1998

Table 1: Vehicles with TK5561 transponder keys.

Licensed under Creative Commons License CC-BY 4.0.
IACR Transactions on Cryptographic Hardware and Embedded Systems Vol. 2018, No.2, pp.1—24, DOI:

der when the vehicle key is inserted. When the driver starts the vehicle, the immobiliser authenticates the transponder before starting the engine, thus preventing hot-wiring.

In this paper, we reverse engineer a widely used vehicle immobiliser system based on the Atmel TK5561 transponder and the AUT64 cipher [Atm06]. The TK5561 transponder is used in a number of Mazda, Ford and Proton vehicle keys, which are shown in Table 1. The TK5561 is based on a patented method of cryptographic authentication [BF03], which uses the AUT64 block cipher and a proprietary authentication protocol.

AUT64 is a 64-bit Feistel network block cipher with a 120-bit secret key. It is used in a number of automotive applications, which include the remote keyless entry system used by most Volkswagen Group vehicles sold between 2004 and 2009 [GOKP16]. AUT64 is round-based and has been found in a number of different configurations, for example, the Volkswagen Group employs AUT64 with 12 rounds. The TK5561 transponder uses AUT64 with either 8 or 24 rounds, depending on a configuration bit set in the transponder's memory.

1.1 Contribution and Outline

The contribution made in this paper is threefold: First, we present the results of reverse engineering AUT64 from a Mazda immobiliser system. We reveal all details of the AUT64 block cipher and the associated TK5561 authentication protocol. Secondly, we present a complete analysis of AUT64, which includes extensive cryptanalysis of the block cipher as a cryptographic primitive in Section 4 and Section 5. We demonstrate the following cryptographic weaknesses in AUT64

- The AUT64 Feistel network compression function is cryptographically weak, as its output is highly predictable;
- Input to the compression function can be precisely controlled by an attacker in the first round of encryption;
- The AUT64 substitution-permutation network behaves non-randomly when its input is nibble-wise symmetric;
- The cryptographically weak output from the first round of encryption can be identified by analysing small sets of ciphertexts;
- The cipher has certain weak keys.

In Section 5.4 and Section 5.5, we identify a novel technique based on integral cryptanalysis that allows us to determine several elements of the secret key. We show that the eight round AUT64 has weaknesses which reduce its key size from 120 bits to no more than 57.5 bits in a chosen-plaintext setting. These results are of independent interest w.r.t. the cryptanalysis of Feistel ciphers with key-dependent permutations and S-Boxes.

Third, in our analysis of the implementation by a major vendor (Section 6), we identify the following weaknesses

1. The key management scheme reduces the permutation key size to just sixteen keys per automotive manufacturer, with one key being used per vehicle family;
2. 32 bits of the key are derived based on the transponder ID, using a manufacturer-wide key derivation function;
3. There are indications that a large part of the key is constant across different vehicles by the same manufacturer.

Weakness	Section	Keyspace (bits)
None	3	120
High-level analysis	3.2	91.5
Permutation Key Size	4.1	88.5
Compression Function Weak Keys	4.2	≤ 87.5
Permutation Weakness	5.1	85.7
Compression Function Weaknesses	5.2, 5.3	≤ 57.7
Integral Cryptanalysis	5.4, 5.5	≤ 78.8
Weaknesses in Key Derivation Scheme	6.1	59.5

Table 2: How each weakness reduces the claimed key space

Based on the cryptographic and implementational weaknesses of AUT64, which we summarise in Table 2, we present two attacks: Our first attack targets the full 24 round AUT64 implementation in the studied system. We show that the security of the system is no more than $2^{48.3}$ bits; and likely much worse in practice, as we have indications that part of the key is constant for many vehicles. Our second attack makes use of the knowledge of 32 bits of the key to break eight rounds of AUT64 within milliseconds using a standard laptop.

1.2 Related Work

The need for transparency into the workings of cryptographic systems has been known since at least the late nineteenth century [Ker83]. Still, there are many examples in the literature of proprietary cryptographic systems which have tried and failed to achieve security by obscurity. Some prominent examples include WEP [SIR02, SSVV14], GSM A5/1 [Gol97, BBK08, PPPM13], DSC for cordless telephones [NTW10, WTHH11], GMR for satellite phones [DHW+12], E0 for Bluetooth [SW06] and CSS for digital rights management [Ste99].

Within the automotive industry, the pitfalls of proprietary cryptography have become apparent. Algorithms originating from the semiconductor industry and specifically marketed as automotive immobiliser and remote keyless entry solutions have been shown to be highly insecure [GOKP16]. The first vehicle immobiliser to be broken was Texas Instruments Digital Signature Transponder (DST) in 2005 [BGS+05]. Since then, two other major cryptographic immobiliser solutions have been reverse engineered and proven to be insecure, namely Hitag2 [VGB12] and Megamos Crypto [VGE15]. Remote keyless entry systems have received a similar treatment. KeeLoq [Bog08, EKM+08] as well as VW Group systems [GOKP16] have been reverse engineered and subsequently found to be insecure. Francillion et al. [FDC10] found that remote key unlocking and vehicle start systems from eight different manufacturers were vulnerable to relay attacks.

AUT64 was first identified by Garcia et al. for its use in a popular Volkswagen Group remote keyless entry system [GOKP16]. Volkwagen's implementation of AUT64 was undermined by the use of a fixed *global master key* and so only a peripheral overview of the cipher was given. In this paper we substantially expand upon this earlier work by reverse engineering, fully specifying and cryptanalysing the AUT64 cipher.

2 Technical Background

In this section we introduce our notation and the cryptographic definitions which are necessary to describe AUT64.

2.1 Notation

Our notation conventions are as follows: when our operands are bytes or nibbles we make use of the operations $\wedge$, $\vee$, $\oplus$, $\ll$ and $\gg$ which are bitwise AND, OR, XOR, left-shift and right-shift, respectively. We use $\|$ to mean concatenation and we use the term *symmetric byte* to refer to a byte b of the form $n \| n$ where $n \in \mathbb{F}_2^4$. When describing byte strings or look up tables, we use square brackets to specify the index. We use the functions msb_m and lsb_m which return the m most and least significant bits, respectively. When our operand is a single byte we use the functions un and ln to mean msb_4 and lsb_4, respectively.

Where $\mathcal{M}$ is some set we use $m \xleftarrow{R} \mathcal{M}$ to denote the assignment to m an element in $\mathcal{M}$ chosen uniformly at random.

2.2 Chosen Plaintext Attack

Security against Chosen Plaintext Attack (CPA) is a standard notion when analysing the security of a cipher. CPA defines an interaction between an adversary and a black-box encryption oracle. The adversary may submit arbitrary messages to the oracle and receive back their encryption. The goal of the attacker is to weaken the security of the cipher used by the oracle.

2.3 Block Ciphers

$\mathcal{K}$ is the key space, $\mathcal{M}$ is the message space and $\mathcal{C}$ is the ciphertext space. Then, a block cipher is defined as a pair of efficiently computable algorithms $\mathbb{E} = (E, D)$, where $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ and $D : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$, such that for all $K \in \mathcal{K}, M \in \mathcal{M} : D(K, E(K, M)) = M$.

2.4 Unbalanced Feistel Networks

A Feistel network is a general method of transforming a function F into a permutation P. Feistel networks are round-based. In each round, they apply a Feistel function F to a subset of the block size n.

The first Feistel network was proposed in the design of Feistel's Lucifer cipher [Sor84]. Lucifer and its descendants, notably the Data Encryption Standard (DES) [DES77] and RC5 [Riv95], are now classified as balanced Feistel networks. For an in-depth review and taxonomy of Feistel networks we refer the reader to [SK96].

Unbalanced Feistel Networks (UFN) are a generalisation of the balanced Feistel network concept shown in Figure 1. A UFN construction relaxes the constraint that the conventionally named left and right halves of the round input must be of equal size, instead they can be independently of any size and are referred to as the target and source blocks.

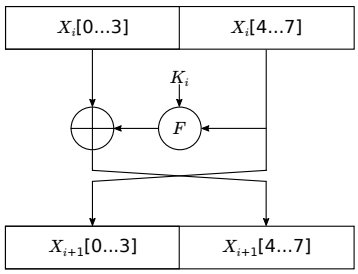


Figure 1: Example of a conventional (Balanced) Feistel construction, where i is the round number.

Generalising yet further, for a construction to be classified as a Feistel network, the only requirement is that one part of the block being encrypted influences the encryption of another part of the block [SK96].

Where s and t are the bit-lengths of the source and target blocks respectively and $s > t$

Definition 1 *We define a fully generalised UFN (GUFN) round to be*

$$X_{i+1} = R\Big(F\Big(k_i, msb_s(X_i), lsb_t(X_i)\Big), msb_s(X_i)\Big)$$

where R is a reversible function and F is reversible in the sense that

$$\forall K, P, Q : F^{-1}(K, P, F(K, P, Q)) = Q$$

In the context of GUFN we introduce the term *cycle length* which is the number of rounds after which every bit in the state has appeared in both the target and source blocks. A balanced Feistel network has a cycle length of two.

3 AUT64

In this section, we present full details of the AUT64 block cipher and the associated TK5561 transponder authentication protocol. The public literature on AUT64 is scant. The cipher is closed source and proprietary, so the only information widely available is from a patent application [BF03] and the product datasheet [TEM98]. To address this problem, we reverse-engineered the complete AUT64 block cipher and its implementation, which we publish here in full detail.

AUT64 was first identified as a proprietary block cipher used in most Volkswagen Group remote keyless entry systems between approximately 2004 and 2009 [GOKP16]. In this paper, we present AUT64 as used by the Atmel TK5561, which is an automotive transponder package for the Atmel e5561 cryptographic IDentification IC (IDIC) [TEM98]. The e5561 uses the AUT64 block cipher and a proprietary authentication protocol to provide a method of authentication for vehicle immobiliser systems. AUT64 is remarkable from a cryptographic design perspective for the fact that, in addition to having a GUFN structure, its symmetric 120-bit key defines a substitution and a permutation from which the cipher's security properties are derived.

In more detail, AUT64 is a 64-bit GUFN block cipher with a key size of 120 bits. In the TK5561, it has either 8 or 24 rounds, depending on the configuration. The AUT64 key space is the triplet of all 32-bit binary strings, all eight-element permutations and all sixteen-element permutations $\mathcal{K} = \langle \mathbb{F}_2^{32}, P_8, P_{16} \rangle$. The 120 bit key size is the sum of the 32, 24 and 64 bits occupied by the $\mathbb{F}_2^{32}$, P_8 and P_{16} key parts respectively.

3.1 Reverse Engineering AUT64

We reverse engineered AUT64 from the Mazda "Module 142" immobiliser system. Concretely, we recovered the firmware from the Motorola MC68HC05B6 microcontroller used in this immobiliser box using a standard programmer. Then, we loaded the firmware binary into the IDA Pro disassembler to perform our analysis. We were able to locate all of the important cryptographic subroutines and reconstruct the AUT64 algorithm and protocol. We implemented a software version of AUT64 and the authentication protocol in both Python and C to develop, test and evaluate our attacks.

We verified our findings against the available e5561 documentation and an implementation that we created for VW remote keyless entry systems [GOKP16]. We found that the immobiliser system uses 24 rounds of AUT64 and the TK5561 authentication protocol. After identifying AUT64 and the TK5561 authentication protocol in the firmware, we were able to locate the permutation and S-Box key parts.

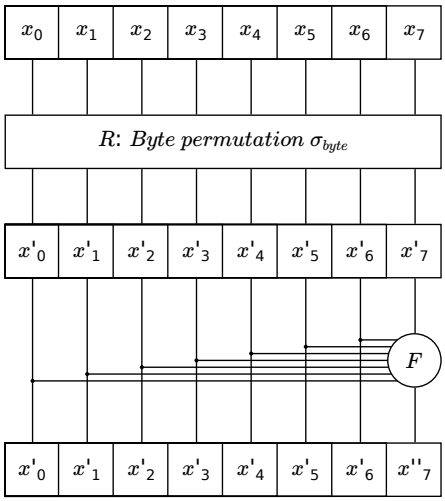
1. The permutation key part k_σ (page 6 in the transponder's memory) is located in the first 32 bits of the microcontroller's programmable read-only memory (PROM) starting at the address `0x0800`. The k_σ we recovered is a cyclic permutation with no fixed points;

2. The substitution key part k_τ (page 8 and 7 in the transponder's memory) is located in the 64 bits which immediately follow the permutation key part. It is bijective, as expected, but otherwise unremarkable.

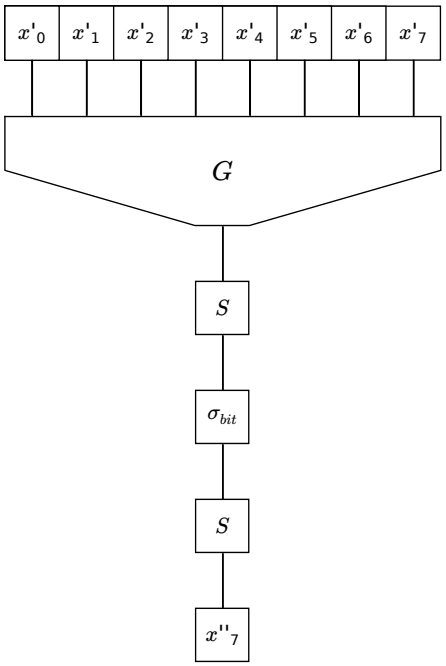
We found that the 32-bit compression function key k_G (page 5 in the transponder's memory) is not stored on the vehicle immobiliser box, instead the base station computes k_G as a function of the transponder's ID (IDcode) that is transmitted at the beginning of the authentication protocol. This is further detailed in Section 6.2.

3.2 Cipher

This section describes the AUT64 cipher in detail. We start by defining the high-level Feistel network, followed by the key specification and then finally describe the full key-dependent structure.



(a) One round of AUT64. The seventh byte of the state x''_7 is changed in each round.



(b) The AUT64 Feistel function F.

Figure 2: The AUT64 Feistel network construction (a) and Feistel function (b).

Definition 2 *(AUT64 cipher state) We define an AUT64 cipher state X as an element in $\mathbb{F}_2^{64}$ where X is composed of eight bytes $x_0, \ldots, x_7$, each an element in $\mathbb{F}_2^8$.*

AUT64 is a GUFN where the source block is the cipher state X and the target block is the byte $x_7 \in X$.

The byte permutation R and Feistel function F are both key-dependent.

Definition 3 *(AUT64 key specification) We define an AUT64 key as a triplet $K \in \langle k_G, k_\sigma, k_\tau \rangle$ where*

1. *$k_G \in \mathbb{F}_2^{32}$ is a 32-bit word from which round keys are derived;*
2. *k_σ is an 8-element permutation which defines both the byte permutation σ_{byte} and the Feistel function bit permutation σ_{bit};*
3. *k_τ is a 16-element permutation which defines the 4×4 bijective S-Box τ in the Feistel function.*

Each round of AUT64, shown in Figure 3a, comprises two components

1. a byte permutation $R(X) = X'$ where

$$R(x_0 \dots x_7) = \Big(\sigma_{\text{byte}}(X)[0], \dots, \sigma_{\text{byte}}(X)[7]\Big)$$

2. a Feistel function $F(X') = x''_7$.

In each round of AUT64, the state is permuted $R : X \to X'$ and then the Feistel function $F : X' \to X''$ is applied. The structure of AUT64 necessitates that, for each bit in the state to appear in both the target and source blocks, it must be applied for a minimum of eight rounds. We can therefore determine that the byte permutation should have a cycle length of eight. If the permutation had any fixed points, then plaintext bits would appear unchanged in the ciphertext. In the remainder of this paper, we assume that the key generation algorithm indeed only chooses permutations with a cycle length of eight.

The TK5561 patent [BF03] specifies a key generation procedure which aligns with the way the AUT64 key is composed. There are three parts

1. The compression function key part k_G is a random bit string generated by means of the DES block cipher which is seeded by the automotive manufacturer. No further information is given in the patent;
2. The permutation key part k_σ is termed the "family key". Each automotive manufacturer is allocated twelve bits, and then selects from sixteen possibilities for the remaining twelve bits;
3. The substitution key part k_τ is termed the "user key". The key is generated using a proprietary method not specified. The patent claims that a repeat will only occur after 20.9×10^{12} user keys have been generated. We can conclude from this claim that the entire 4×4 S-Box space is utilised.

Definition 4 *(AUT64 Feistel function) The AUT64 Feistel function F is constructed from the following four components*

1. *A compression function $G(X') = g$ which maps a permuted input state $X' \in \mathbb{F}_2^{64}$ to an output byte $g \in \mathbb{F}_2^8$;*
2. *A substitution operation S*

$$S(g) = \tau\Big(un(g)\Big) \parallel \tau\Big(ln(g)\Big)$$

which comprises two identical 4×4 S-Boxes τ applied independently to the upper and lower nibble output of G;

3. *A permutation operation*

$$\sigma_{bit}\Big(S(g)\Big)$$

which applies the same transposition as σ_{byte} but bitwise to the output of $S(g)$;

4. *The substitution operation applied to the output of the bitwise permutation operation*

$$x''_7 = S\left(\sigma_{bit}\Big(S(g)\Big)\right)$$

The final three operations form a Substitution-Permutation Network (SPN).

In the following, we outline the AUT64 round key scheduling mechanism. A unique round key is derived from the compression function key part k_G in each round. The key schedule is derived and applied within the compression function as shown in Figure 4. AUT64 round keys are a tuple of permutations of the compression function key. The AUT64 key schedule divides k_G into eight nibbles. Two key schedule tables T_U and T_L (see Figure 7c in the appendix) prescribe a permutation for each round and are applied to k_G to derive a round key for the upper and lower nibbles in the input state, respectively.

Definition 5 *(AUT64 round keys) When r is the round number and i the byte index in the permuted round state X', we define*

$$uk(k_G, r, i) = k_G\left[T_U\left[(r \times 8) + i\right]\right]$$

$$lk(k_G, r, i) = k_G\left[T_L\left[(r \times 8) + i\right]\right]$$

which return the lower and upper round key nibble, respectively.

The key schedule tables T_U and T_L introduced in Definition 5 have the following specifications

- They are non-identical key-independent look up tables which define the nibble-wise key schedule for each round of AUT64. Each table comprises $8 \times N$ elements, where N is the number of AUT64 rounds;
- Both tables are divided into blocks of $8 \times N$, which correspond to the AUT64 cycle length. XORing the two tables together reveals a constant difference between the corresponding round key nibbles in each $8 \times N$ block;
- The two tables can be merged nibble-wise to yield a $16 \times N$ round key schedule. Assembling the key schedule in this way reveals that there are no repeated round keys.

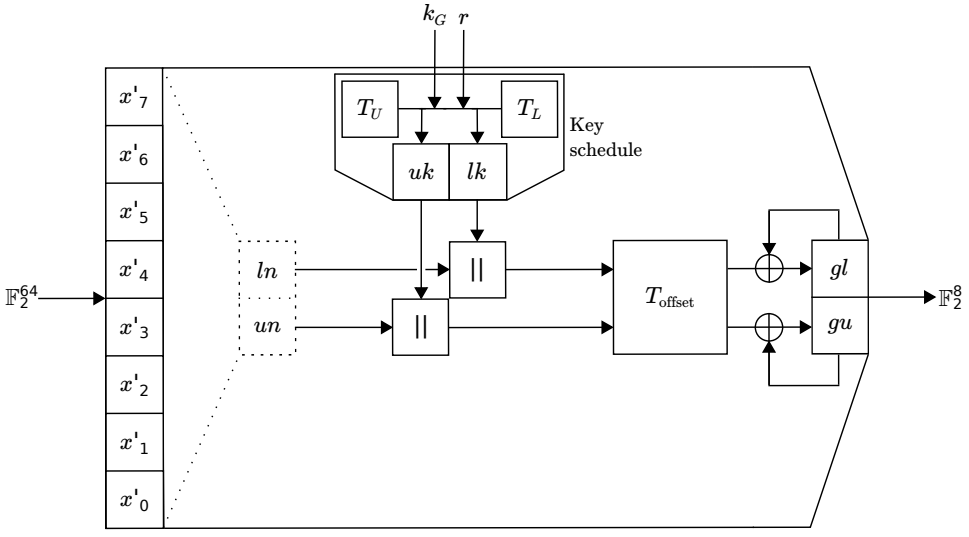


Figure 4: The AUT64 compression function G.

Definition 6 *(AUT64 compression function) The AUT64 compression function G takes as input the key part k_G, the permuted block state X' and the round number r. G outputs the concatenation of two internal variables which are calculated as follows*

$$gl = \bigoplus_{i=0}^{7} T_{offset}\Big[lk(k_G, r, i) \parallel ln(X'_i)\Big]$$

$$gu = \bigoplus_{i=0}^{7} T_{offset}\Big[uk(k_G, r, i) \parallel un(X'_i)\Big]$$

The compression function look up table T_{offset} (Figure 7b in the appendix) has the following properties

- It contains 256 elements, is key-independent and is best conceptualised as a 16×16 array of nibble values. The compression function selects elements from the table using a round key nibble to select a column and a state nibble to select a row;
- It is symmetric about its descending diagonal axis such that $\forall u, v \in \mathbb{F}_2^4$

$$T_{\text{offset}}[u \parallel v] = T_{\text{offset}}[v \parallel u]$$

- The first row and first column contain only the value zero. With reference to Definition 6, this means that for any nibble in the input state X' with the value zero, T_{offset} will always return zero. Similarly if a round key nibble has the value zero, then T_{offset} will always return zero regardless of the relevant input state nibble value.

3.3 Authentication Protocol

This section describes the TK5561 authentication protocol. Authentication takes place between a vehicle immobiliser box B and a transponder key T. The design goals of the protocol stated in the product patent are to provide a method for authentication and to prevent known and chosen plaintext cryptographic attacks [BF03].

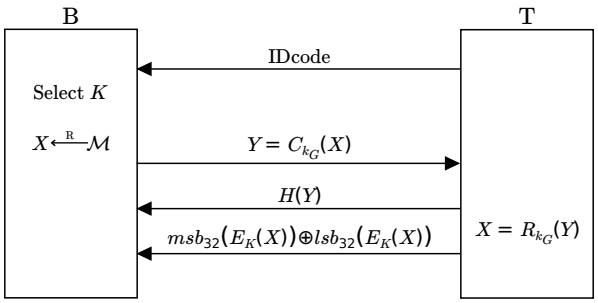


Figure 5: The TK5561 authentication protocol.

Definition 7 *(AUT64 authentication) We define AUT64 authentication over a keyspace $\mathcal{K} = \mathbb{F}_2^{32}$ and a message space $\mathcal{M} = \mathbb{F}_2^{64}$ as a quartet of algorithms $\mathbb{A} = (E, C, R, H)$ where for all $K \in \mathcal{K}$ and $X, Y \in \mathcal{M}$*

1. *$E_K : X \mapsto E(K, X)$ is the AUT64 block cipher;*
2. *$C_{k_G} : Y \mapsto C(k_G, X)$ is a challenge algorithm which is keyed with the AUT64 compression function key part $k_G \in K$;*

3. $R_{k_G} : X \mapsto R(k_G, Y)$ *is a keyed response algorithm such that* $R\Big(k_G, C(k_G, X)\Big) = X$;

4. $H(Y)$ *returns the Hamming weight of* Y.

The TK5561 authentication protocol shown in Figure 5 has four parts:

1. The transponder key T initiates the protocol by sending the pre-shared public value *IDcode* by which the vehicle immobiliser box B determines which symmetric AUT64 key K to use for the remainder of the protocol. One immobiliser box may have the keys for several different transponder keys;

2. The vehicle immobiliser box B generates the nonce $X \xleftarrow{R} \mathcal{M}$ and then computes the challenge by applying the proprietary challenge algorithm $Y = C_{k_G}(X)$. The resulting challenge Y is sent to the transponder key T;

3. The transponder key T receives the challenge and then recovers the original nonce by applying the response algorithm $X = R_{k_G}(Y)$. The transponder key confirms to the immobiliser box B that it has received the correct challenge by transmitting the Hamming weight of the challenge $H(Y)$;

4. The vehicle immobiliser box B and the transponder key T both compute the AUT64 encryption of the nonce $E_K(X)$ and then XOR together the top and bottom halves of the ciphertext. The transponder sends the 32-bit result back to the immobiliser. If the values match, the vehicle ignition is enabled.

The proprietary nonce encryption algorithm C_{k_G} is a Linear Feedback Shift Register (LFSR) based stream cipher, which is seeded with the 32-bit part k_G of the complete AUT64 key XORed with a constant. More precisely, the challenge encryption algorithm has a 32-bit internal state $Z \in \mathbb{F}_2^{32}$ that is seeded such that:

$$Z = k_G[0] \oplus \texttt{0xD5} \parallel k_G[1] \oplus \texttt{0x89} \parallel k_G[2] \oplus \texttt{0x0C} \parallel k_G[3] \oplus \texttt{0x7B}$$

For each bit in the keystream, the algorithm computes a byte derived from the LFSR state:

$$Z' = Z[3] \oplus \Bigg(Z[0] \oplus \Big(Z[0] \oplus (Z[0] \gg 4)\Big) \gg 1\Bigg) \gg 1$$

The LFSR shifts right, and the rightmost bit of Z' is fed back into Z from the left into bit position 0. The keystream is read from the LFSR state Z at bit 0.

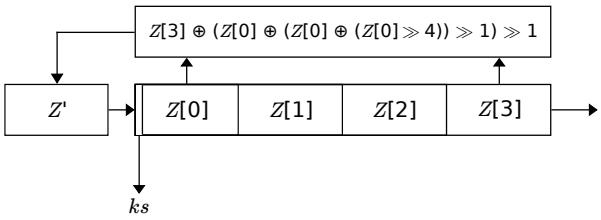


Figure 6: Authentication challenge algorithm LFSR.

Each time the challenge algorithm is applied 64 bits of keystream are generated by the challenge algorithm LFSR. The keystream is then XORed with the session nonce X to produce the challenge value Y.

In the remainder of this paper, we first point out a series of cryptographic weaknesses in the design of AUT64, that apply when the algorithm is used as a normal block cipher. Then, in Section 6 we focus on a concrete implementation and present a full attack on the respective usage of AUT64 within the immobiliser.

4 AUT64 Weak Keys

In this section, we analyse AUT64 and identify a number of weak keys. We define a weak key to be any AUT64 key which makes the cipher behave in an undesirable way. The AUT64 documentation states that a key is 120 bits long. A more in-depth analysis (see Section 3.2) reveals that whilst the key occupies 120 bits, it only has approximately 91.5 bits of entropy. This discrepancy arises because only 32 bits of the key define a bit string k_G. The remaining 88 bits define a bijective 4×4 S-Box k_τ and an 8-element permutation k_σ. Bijective S-Boxes and permutations both express an arrangement of discrete elements with no repetitions. The condition that no two elements can have the same value is the source of reduced entropy in the key space.

4.1 Permutation Key Size

In this section we show that the permutation key part k_σ size is reduced as a consequence of the AUT64 Feistel structure.

The cycle length of AUT64 is determined by the byte permutation σ_{byte}. In order for each bit of the plaintext to influence each bit in the ciphertext we require that each byte must be moved to x_7 and the Feistel function applied. The byte permutation must therefore be circular (cyclic with no fixed points) and this reduces the entropy of the permutation key part. The number of n bit cyclic permutations is $(n-1)!$ therefore the reduced σ_{byte} key size is $(8-1)! \approx 2^{12.3}$ bits.

Any known permutation elements, such as those revealed by bytes which are unchanged from the plaintext due to non-cyclic permutation keys, or by the observation that all of the bytes are changed, further reduce the key size of k_σ. If k_σ is cyclic and n elements become known, the remaining key size is $|k_\sigma| = (7-n)!$ [Bru10].

4.2 Compression Function Weak Keys

We analysed the AUT64 compression function G and identified a set of weak keys. We know from Definition 6 that the output of G is an XOR sum of values from the lookup table T_{offset}. For any input state nibble with the value zero, T_{offset} always returns zero. Similarly, if a round key nibble has the value zero, then T_{offset} always returns zero, regardless of the relevant state nibble.

If AUT64 is used in an application which requires decryption, then the compression function key part k_G can never contain any key nibbles with the value zero. Decryption requires that the original plaintext can be recovered from the ciphertext and its intermediate states. This is only possible when the T_{offset} column index can be determined. Any k_G nibble with the value zero will select the first row of T_{offset} and output the value zero for all column indices selected by the corresponding state nibble. Since the round output does not encode the column index, the round input cannot be decrypted and AUT64 is no longer bijective.

More generally, if k_G contains nibbles with the value zero, then the security of AUT64 is reduced and the compression function behaves non-injectively. In the most extreme case where all of the nibbles in k_G are zero, $G(X')$ is reduced to the constant zero. Removing the keys in k_G which contain zero nibbles reduces the key size $|k_G|$ to $15^8 \approx 2^{31.3}$.

If AUT64 is used for eight rounds, then the compression function key nibbles k_{G_3} and k_{G_6} are of particular significance to the security of the cipher. It is possible to determine whether either key nibble has the value zero and whether they have the same value by a chosen plaintext attack on the block cipher.

A chosen plaintext attack for determining if $k_{G_3} = 0$, $k_{G_6} = 0$ and $k_{G_3} = k_{G_6}$ is as follows

1. Submit the plaintext $(\texttt{0x00})^8$ to the oracle and record the corresponding ciphertext $c_{\text{ref}} = E_K\big((\texttt{0x00})^8\big)$;
2. Generate a set of eight plaintexts to determine if $k_{G_3} = \texttt{0}$. Each plaintext should contain seven bytes with the value 0x00 and one byte with a constant nibble value n in the lower position of the byte 0x0n. The non-zero byte should be in a different byte position for each plaintext;
3. Submit the eight plaintexts to the oracle. Compare each resulting ciphertext to c_{ref};
4. If one of the ciphertexts matches the encryption of the first plaintext then $k_{G_3} = 0$;
5. Repeat the test for $k_{G_6} = \texttt{0}$. Generate another eight plaintexts but place the constant nibble value n in the upper position of the non-zero byte 0xn0;
6. If one of the ciphertexts matches the encryption of the first plaintext then $k_{G_6} = \texttt{0}$.
7. To determine if $k_{G_3} = k_{G_6}$, compare the two sets of ciphertexts. Check whether there is a pair of ciphertexts which both contain a byte at the same position such that the lower nibble, in the lower nibble ciphertext set, is equal to the upper nibble, in the upper nibble ciphertext set.

If a weak key is revealed in this attack, it reduces the key size of k_G by either four or eight bits, depending on whether one or both key nibbles have the value zero. If neither key nibble has the value zero but they are determined to be equal, the key size is reduced by approximately four bits.

5 AUT64 Cryptanalysis

In this section, we present our cryptanalysis of AUT64. We identify a number of weaknesses and derive corresponding attacks. Typically, a block cipher would be attacked by applying linear [Mat94] or differential cryptanalysis [BS91]. The key-dependence of the permutation and S-Box used in AUT64 led us to seek alternative cryptanalytic techniques which provide more general attacks independent of the key. Our cryptanalysis is predominantly applicable to the eight round AUT64 configuration, and treats AUT64 as a general block cipher in a chosen-plaintext setting.

5.1 Permutation Weakness

In this section, we present four weaknesses of the AUT64 block cipher, which we use to determine an element of the permutation key part k_σ. We present a sixteen chosen plaintext attack which reduces the cyclic permutation key-part to a size of $|k_\sigma| = (7-1)! \approx 2^{9.5}$. The four weaknesses which we use to reduce the security of AUT64 are

1. The compression function G is cryptographically weak. Careful manipulation of its input can cause outputs which are highly distinguishable;
2. In the first round of AUT64, input to the compression function can be tightly controlled by an attacker;
3. The SPN which takes the output of the compression function and completes the Feistel function F has a cryptographic weakness;
4. After eight rounds of AUT64, the output from the first round Feistel function is a byte in the ciphertext.

We present a sixteen chosen plaintext attack which reveals one element of the permutation key-part. The chosen plaintexts for this attack are the set of plaintexts such that each plaintext contains eight identical and symmetric bytes

$$\mathbb{P} = \left\{ (n \parallel n)^8 : n \in \{0, \ldots, 15\} \right\}$$

Each plaintext P in $\mathbb{P}$ manipulates the compression function G such that it always outputs a symmetric byte in the first round. For each plaintext $P_n \in \mathbb{P}$ the attack works as follows

1. The byte permutation σ_{byte} has no effect since all of the bytes in the plaintext have the same value. This allows the input to the compression function to be carefully controlled.

2. The unchanged plaintext is input to the compression function $G_{k_G}(P_n)$ which unconditionally outputs a symmetric byte $p'_n \parallel p'_n$. This property holds because the compression function yields an XOR sum of values derived from each byte in the input (see Definition 6). If all the input bytes have the same value, the output becomes the XOR sum of up to eight values from a single column in T_{offset}. Since the key schedule (Definition 5) prescribes permutations of the same underlying key k_G in each round, the nibble-wise XOR sums output by G will have the same final value. The order in which the terms in the sum are added will differ, but the set of terms and the final value will always be equal.

3. The symmetric byte $p'_n \parallel p'_n$ is input to the SPN which completes the AUT64 Feistel function F (see Definition 4).

The Feistel function output byte does not generally retain the nibble-wise symmetry of $p'_n \parallel p'_n$ because the bitwise permutation σ_{bit}, on average, removes the symmetry. There are however at least two σ_{bit} inputs in which the symmetry property is unconditionally preserved. When the input is either `0x00` or `0xFF`, σ_{bit} has no effect because all the input bits are equal.

We exploit the property that bytes $x_0 \ldots x_6$ are fixed during each round, under this condition the compression function G is bijective with respect to the seventh byte x_7. Each of the sixteen chosen plaintexts in $\mathbb{P}$ cause G to output a symmetric byte, and since we require decryption and therefore to be able to determine a unique seventh input byte in each round, we know that $\mathbb{P}$ causes all possible sixteen symmetric bytes $p'_n \parallel p'_n$ to be input to the SPN.

Since the substitution operation T is also bijective, it is guaranteed that both `0x00` and `0xFF` are input to σ_{bit}. When the set of chosen plaintexts $\mathbb{P}$ is encrypted, the outputs of the Feistel function in the first round will always contain at least two symmetric bytes.

Once all sixteen chosen plaintexts have been encrypted, the first round of AUT64 can be distinguished by identifying the byte index which contains the symmetric bytes from the first round in the set of ciphertexts $\mathbb{C}$, as follows

1. Define lists $\mathbb{C}'_i$ of ciphertext bytes for each of the eight byte indices $i \in \{0, \ldots, 7\}$;

2. From each ciphertext $C_n \in \mathbb{C}$ add the byte at the index i corresponding to the list $\mathbb{C}'_i$;

3. Count the number of symmetric bytes in each list.

The output from the first round of AUT64 is the list index which contains the greatest number of symmetric bytes. The list corresponding to the first round always contains at least two symmetric bytes (and often more). We analyse the key-dependent distribution of symmetric bytes output under this attack in Section 5.2. Identifying the first round permutation element reduces the permutation key entropy from $|k_\sigma| = 7! \approx 2^{12.3}$ to $(7-1)! \approx 2^{9.5}$.

5.2 Compression Function Symmetric Bytes

Now, we identify and exploit a cryptographic weakness of the compression function and the Feistel function when handling symmetric bytes to reduce the total AUT64 security to no more than $|K| \approx 2^{76.9}$. For a class of weak keys, we show that the security is reduced to just $2^{44.7}$.

When the set of symmetric byte chosen plaintexts $\mathbb{P}$ from Section 5.1 is encrypted with AUT64, the symmetric compression function G_{k_G} output bytes in the first round belong to an equivalence class. There are sixteen different equivalence classes. Each key can be classified by the XOR sum of its nibbles

$$k_{G_{class}} = \bigoplus_{i=0}^{7} k_{G_i}$$

The G_{k_G} output sets which characterise each equivalence class can be read from k_{offset}. The compression function classifier $k_{G_{class}}$ selects a row from the table and the chosen plaintext nibble value selects a column. Analysis of the compression function output in the first round under chosen plaintexts $\mathbb{P}$ reveals two distributions

1. A degenerate distribution where all elements in the output have the same value. When $k_{G_{class}} = \texttt{0}$, the XOR sum of T_{offset} values is also equal to zero. One in every sixteen keys in k_G has this weakness and reduces the key size accordingly to $(\frac{1}{16} \times 2^{32}) = 2^{28}$. If we only include non-weak k_G keys (see Section 4.2), then the key size is further reduced to approximately $(\frac{1}{16} \times 15^8) \approx 2^{27.3}$;

2. A uniform distribution, where there is one of each element in the output, is the most likely outcome. This reduces the compression function key size for non-weak keys to $(\frac{15}{16} \times 15^8) \approx 2^{31.2}$.

The compression function output distribution is preserved by the SPN which follows G and completes the Feistel function F (see Figure 3b). Analysis of the Feistel function output in the first round reveals information about the substitution key part k_σ. There are two different ciphertext byte list C'_i distributions to consider

1. A degenerate byte distribution; all output bytes have the same value:

$$\forall c'_i \in C'_i : c'_i = T\left(\sigma_{\text{bit}}\Big(S(\texttt{0x00})\Big)\right)$$

2. A uniform distribution, in which there are sixteen different output bytes, at least two of which are symmetric.

When C'_i is a uniform distribution and contains exactly two symmetric bytes, they reveal the substitutions of the values `0x0` and `0xF`. These values are unaffected by any σ_{bit} permutation and guarantee that the output of the Feistel function is a symmetric byte. We analysed all possible circular permutation key values k_σ applied to the set of all sixteen symmetric bytes which may be output by G_{k_G} under $\mathbb{P}$. We found the following distinguisher on the permutation key part k_σ[1]: 46%, 42%, 11% and 1% of k_σ keys yield exactly two, four, eight and sixteen symmetric bytes, respectively, at the output of the compression function G.

On average C'_i, contains exactly two symmetric bytes and the S-Box key size is reduced from $|k_\tau| = 16! \approx 2^{44.3}$, to $\leq 2 \times 14! \approx 2^{37.3}$. The permutation key size is also reduced to $0.46 \times (7-1)! \approx 2^{8.4}$ and $|k_G|$ to approximately $2^{31.2}$. In total the AUT64 key size is reduced to no more than $|K| \approx 2^{76.9}$.

[1]Key size percentages given are approximate. The specific values are 2304, 2112, 576 and 48 keys for two, four, eight and sixteen symmetric bytes in C'_i respectively.

The best case for an attacker is that $\mathbb{P}$ yields sixteen symmetric bytes in C'_i. This outcome reduces $|k_\sigma|$ to $48 \approx 2^{5.6}$, $|k_\tau|$ to $15 \times 16 \approx 2^{7.9}$ (16 possibilities for each class in $k_{F_{\text{class}}}$) and $|k_G|$ to $\approx 2^{31.2}$. For weak k_σ, the resulting effective key size for eight round AUT64 is $|K| = |k_\sigma| \times |k_\tau| \times |k_G| \approx 2^{44.7}$. Sometimes C'_i may not be readily identifiable by counting symmetric byte elements. This is usually because one or more other lists in $\mathbb{C}'$ also contain an unusual number of symmetric bytes. For this case, we have developed a second technique.

First, divide each list in $\mathbb{C}'$ into an upper and lower nibble list and then test them for equality. The first round list is distinguishable by an identical set of values in the upper and lower nibble lists.

There is one final exception, which is in the case that k_G is weak such that G_{k_G} is not bijective (see Section 4.1). If G_{k_G} is not bijective, then the first round can be distinguished using a 256 chosen plaintext attack based on integral cryptanalysis as detailed in Section 5.4.

5.3 Compression Function Divide-and-Conquer

In this section, we present a second weakness in the AUT64 compression function, which is caused by the way the function uses its key part k_G. We present a 32 chosen plaintext attack which exploits this weakness to reduce the AUT64 key size to no more than $2^{57.7}$. In each round, G outputs two nibblewise XOR sums of values $g_u \parallel g_l$ from the lookup table T_{offset} (see Definition 6). The key usage of G and the properties of T_{offset} can be exploited to perform a divide-and-conquer attack on the compression function key part k_G. We present a chosen plaintext attack which causes the output of the Feistel function in the first round to be dependent on only one nibble in k_G

1. Identify the ciphertext byte position corresponding to the first round r_0 by performing the sixteen chosen plaintext attack described in Section 5.1;
2. Generate the set of sixteen chosen plaintexts defined nibblewise such that each plaintext contains fifteen nibbles with the value zero and a 4-bit counter value
$$\mathbb{P} = \left\{ \left(n \ll (64 - 8 \times r_0) \right) : n \in \{0, \ldots, 15\} \right\}$$
3. Submit the chosen plaintexts to the oracle and record the corresponding ciphertext bytes at position r_0 in the list $\mathbb{C}'_{r_0}$;
4. Exhaustively search the reduced key space. For each possible k_τ, k_σ and each k_{G_3} nibble value, encrypt offline the set of chosen plaintexts $\mathbb{P}$. For each plaintext encrypted under each key, test whether the ciphertext byte at position r_0 matches the corresponding reference value in $\mathbb{C}'_{r_0}$. Once all sixteen ciphertext bytes match those produced by the oracle, the correct K'_{G_3} nibble, as well as the k_σ and k_τ key parts have been identified;
5. The remaining seven k_G key nibbles can be efficiently determined by supplying additional chosen plaintexts to the oracle that cause outputs dependent on the remaining key nibbles. Since k_σ and k_τ are known, each key nibble will only require sixteen chosen plaintexts and up to sixteen offline encryptions.

This attack requires 32 chosen plaintexts and reduces the security of AUT64 to $(16! \times 6! \times 15) + (16 \times 7) \approx 2^{57.7}$ encryptions. The security is the product of the k_τ, k_σ and k_{G_3} key sizes and sixteen offline encryptions. Most keys will fail to match even the first byte in the reference set $\mathbb{C}'_{r_0}$ and so the plaintext encryptions for each key can be reduced to an average of one offline encryption. $2^{57.7}$ encryptions is considerably less security than the AUT64 ideal key entropy of 91.5 bits, and is within the reach of specialised setups for exhaustive key search [GKN+08]. The attack can be performed offline once the initial 32 chosen plaintexts have been encrypted.

5.4 Integral Cryptanalysis

Next, we show how the weaknesses we identified in Section 5.1 can be generalised to perform a divide-and-conquer chosen plaintext attack using techniques from integral cryptanalysis to reveal an element from the permutation key part k_σ. Concretely, we identify a 256 chosen plaintext attack which reduces the cyclic permutation key size from $|k_\sigma| = 7! \approx 2^{15.3}$ to $(8-3)! \approx 2^{6.9}$.

Integral cryptanalysis is a technique with similarities to differential cryptanalysis. It is particularly effective against block ciphers which use only bijective components [KW02]. Where differential cryptanalysis typically considers plaintext-ciphertext pairs with a constant XOR difference, integral cryptanalysis uses sets of chosen plaintexts and their corresponding encryptions with a constant difference over the entire sets. Eight round AUT64 is vulnerable to integral cryptanalysis where the plaintexts are constructed adaptively depending on the cipher behaviour. We propose the following method for distinguishing the second round byte index of AUT64 using integral cryptanalysis

1. Generate a set of 256 plaintexts such that in each plaintext, seven bytes have the value `0x00` and the eighth byte, placed at the byte index corresponding to the first round, is an 8-bit counter which has a unique value in each plaintext

$$\mathbb{P} = \left\{ \left(n \parallel (\texttt{0x00})^7 \right) : n \in \{0, \ldots, 255\} \right\}$$

2. Encrypt all of the plaintexts and store the corresponding set of ciphertexts $\mathbb{C}$;

3. Build $\mathbb{C}'$ from $\mathbb{C}$ by taking each byte in each ciphertext and adding it into a set $C'_i \in \mathbb{C}'$ according to its byte index i;

4. Compute the XOR sum over each set in $\mathbb{C}'$

$$\forall C'_i \in \mathbb{C}', \forall c_j \in C'_i : C'_{i_{\text{sum}}} = \bigoplus_{j=0}^{255} c'_j$$

This method identifies two list integrals $C'_{i_{\text{sum}}}$ with the value zero. One corresponds to (and confirms) the first round byte index, the second identifies the byte index position, which corresponds to the second round of encryption. An intuition for why this attack works is to consider that because AUT64 is bijective, each round must be bijective; therefore it is necessary for each plaintext in $\mathbb{P}$ to encrypt to a unique ciphertext byte in the first round. If this were not the case, then the original plaintext could never be recovered from the intermediate state caused by the output of the first round. The method we propose causes the output from the first two rounds to each be a permutation of the set of all possible byte values. For each plaintext $P_n \in \mathbb{P}$ the attack works as follows

1. The byte permutation σ_{byte} applied to P_n yields a transposed plaintext P'_n of the same form: seven bytes with value `0x00` and one unique counter byte. Since it has been placed at the byte index which corresponds to the first round, the counter is always moved to byte position seven in P'_n;

2. Where j is the byte counter, the Feistel function F applied to the transposed plaintext yields an encrypted intermediate state

$$X' = (\texttt{0x00})^7 \parallel F_K\left((\texttt{0x00})^7 \parallel j \right)$$

3. The byte permutation is applied to the intermediate state X' and the encrypted byte from the first round j' is transposed to a byte index position determined by the permutation key part k_σ;

4. The Feistel function is applied to the permuted intermediate state. Each permuted intermediate state $\sigma_{\text{byte}}(X')$ has seven bytes with value `0x00` and one, at an unknown index, with the encrypted counter value j';

5. The output from the second round of encryption $F_K\left(\sigma_{\text{byte}}(X')\right)$ is written to the cipher state and then encryption continues for another six rounds.

After eight rounds of AUT64, each ciphertext contains a byte from the first round round j' and another from the second round j''. In both rounds, the input to the compression function has only one non-zero byte. We can be certain that the output ciphertext byte lists $C'_{r_0}, C'_{r_1} \in \mathbb{C}'$ both contain the entire uniform set of all 256 byte values. The integral XOR sums $C'_{r_{0_{\text{sum}}}}$ and $C'_{r_{1_{\text{sum}}}}$ are zero, and the first and second byte indices are identified. In Figure 8 in the appendix we show the histogram representation of this attack.

This weakness reduces the permutation key part by one additional known permutation element. Applying this attack, its prior in Section 5.1, and determining that k_σ is cyclic (see Section 4.1) requires a total of 272 chosen plaintexts and reduces the cyclic permutation key size from $|k_\sigma| = 7! \approx 2^{12.3}$ to $(7-2)! \approx 2^{6.9}$.

5.5 Extending the Integral Cryptanalytic Method

Finally, we generalise the integral cryptanalytic method from Section 5.4 and determine its limitations. We present a chosen plaintext attack for determining the third and fourth round permutation key parts and show that the permutation key size $|k_\sigma|$ is reduced to just six possible keys. Permutation elements of an AUT64 key can be iteratively identified by adaptively formulating chosen plaintext attacks which propagate an integral sum distinguisher into a target round.

The type of chosen plaintexts necessary for identifying the third round are those with six `0x00` bytes and two counters n, m placed at the byte positions corresponding to round one and two, respectively. For example, if round one and two are at byte indices zero and seven respectively

$$\mathbb{P}_{nm} = \left\{ (n \parallel (\texttt{0x00})^6 \parallel m) : n, m \in \{0, \ldots, 255\} \right\}$$

From the set of 2^{16} plaintexts $\mathbb{P}_{nm}$ the ones which determine the third round are the subset which result in intermediate states such that the encrypted output bytes corresponding to rounds one and two are equivalent: $n' = m'$.

A general method for building the set of plaintexts $\mathbb{P} \subset \mathbb{P}_{nm}$ which cause the integral sum $C'_{r_{2_{\text{sum}}}} = 0$ is to exhaustively encrypt the set of plaintexts $\mathbb{P}_{nm}$. Check each ciphertext for equality at the byte indices belonging to rounds one and two. When $n' = m'$, add the corresponding plaintext to the attack set $\mathbb{P}$.

For each chosen plaintext $P_n \in \mathbb{P}$, the internal state after two rounds of encryption is two equal encrypted counter bytes and six bytes with the value `0x00`. The bijective Feistel function property determines that under these conditions, the third round will always output a unique ciphertext byte for each unique encrypted counter pair.

Now, we compute the XOR sum over each ciphertext byte-index list $C'_i \in \mathbb{C}'$. The lists corresponding to the indices of rounds one, two and three all have an XOR sum of zero. Subtracting the known indices of the first and second rounds uniquely identifies the permutation element corresponding to the third round of encryption.

Continuing our attack, the fourth round can be identified by applying the same method, but placing three byte counters in the exhaustive set of plaintexts at the indices corresponding to rounds one, two and three. Encrypt all 2^{24} plaintexts and test for equality of the three encrypted counter values. Add the plaintexts which meet the ciphertext equality condition into a chosen plaintext set $\mathbb{P}$, which is then used to identify the fourth round by computing the corresponding ciphertext byte index list sums.

Determining the ciphertext byte indices of the first three rounds of AUT64 requires a total of $528 + 2^{16}$ chosen plaintexts and reduces the cyclic permutation key size from $|k_\sigma| = 7! \approx 2^{12.3}$ to $(7-3)! = 24 \approx 2^{4.6}$. Determining the first four rounds requires a total of $784 + 2^{16} + 2^{24} \approx 2^{24}$ chosen plaintexts and reduces the cyclic permutation key size from $|k_\sigma| = 7! \approx 2^{12.3}$ to $(7-4)! = 6$.

The limitation of this weakness is that the number of chosen plaintext encryptions grows exponentially in the number of permutation elements which are determined. It is not efficient to continue the attack beyond identifying the first two rounds. The most optimal strategy is to use the weakness of Section 5.4 to reduce the permutation key size to 120 keys using 272 chosen plaintexts, and to search the remaining keys offline.

5.6 More than Eight Rounds

The cryptographic weaknesses we have presented are all dependent on an adversary being able to distinguish the output of the Feistel function in the first round. If AUT64 is implemented with more than eight rounds, then this property is negated. A ninth round of AUT64 will always overwrite the byte output by the first round.

In general, we are able to distinguish $16 - N$ rounds from N round AUT64 by the application of the attacks in Section 5.4. The VW remote keyless entry system uses a twelve round implementation of AUT64. Our methods are therefore able to distinguish $16 - 2 = 4$ permutation key parts in this implementation. Attacking twelve rounds of AUT64 requires resolving uncertainty as to which byte indices the counters corresponding to the positions of the first four rounds need to be placed.

To distinguish the fifth round, four simultaneous counter positions must be identified from eight possible locations. Therefore in twelve round AUT64, $\binom{8}{4} \times 2^{32} \approx 2^{38.1}$ chosen plaintexts are required to distinguish the output of the fifth round. This attack reduces $|k_\sigma|$ from $7! \approx 2^{12.3}$ to $4! \times 2! \approx 2^{3.6}$.

6 Attacking a Concrete System using AUT64

In this section, we present our results for attacks on the implementation of AUT64 in a real immobiliser system. We first outline the utilised scheme for key derivation, and present a corresponding attack that breaks AUT64 in the 24 round setting (as configured by the vendor). We also outline a very fast attack on 8-round AUT64 when k_G is known.

6.1 Weaknesses in Key Derivation Scheme

Based on our reverse engineering, we found that the immobiliser box derives the 32-bit key part k_G as a function of the (public) IDCode of the transponder. More precisely, let ID_0, ID_1, ID_2, ID_3 represent the four bytes of IDCode (page 1 in the transponder's memory).

Definition 8 *(k_G key derivation) Further, let $u = (ID_0 \wedge \texttt{0xE}) \ll 1$, T_D be a 32 byte key-independent look up table. Then k_G is defined bytewise as follows*

$$k_{G_3} = ID_3 \oplus T_D[3+u]$$
$$k_{G_2} = ID_2 \oplus T_D[2+u] \oplus k_{G3}$$
$$k_{G_1} = ID_1 \oplus T_D[1+u] \oplus k_{G3}$$
$$k_{G_0} = ID_0 \oplus T_D[u] \oplus k_{G3}$$

T_D is given in Figure 7a in the appendix. The use of a key derivation function means that an adversary can trivially recover k_G with a single communication to the transponder (to obtain the IDCode). The remaining task is thus to determine the permutation and the S-Box.

6.2 Attacking a 24-Round AUT64 Implementation

The studied immobiliser systems uses AUT64 in the 24-round configuration. To recover k_τ and k_σ, we first obtain k_G based on capturing the transponder's IDCode. The anticipated remaining key size is $|K| = 16! \times 7! \approx 2^{56.5}$ (assuming all 4×4 S-Boxes and all cyclic permutations are possible), which is already in the range of dedicated brute force devices such as [GKN+08]. However, a property of the key management scheme specified in the AUT64 patent is that it leaves only sixteen possible permutation keys (the so-called "family key") per vehicle manufacturer. That means that by reading k_σ from two different immobiliser boxes and identifying the constant part (this is a one-time process), the remaining AUT64 key space is only $|K| = 16! \times 16 \approx 2^{48.3}$.

Further, we have indications that the studied system actually uses a constant k_τ and k_σ for a range of vehicles. More precisely, we recovered k_τ and k_σ from two different immobiliser boxes and obtained the same values. If this assumption can be confirmed, it implies that the full key of the transponder can be recovered solely by using the knowledge of IDCode (to derive k_G) and the manufacturer-wide values k_τ and k_σ, without requiring further cryptanalysis. We leave the confirmation of this conjecture for future work.

6.3 Attacking Eight Round AUT64

Finally, we give an example for exploiting the cryptographic weaknesses of Section 5 in a practical setting. We assume that (i) AUT64 is used with 8 rounds, (ii) k_G can be predicted (e.g. if the derivation function described above is used), and (iii) that we obtain all 8 output bytes. Note that in the TK5561 protocol, we obtain the XOR of the two 32-bit halves, while the patent specifies this reduction to 4 byte as optional. We leave the extension of the following attack to the case when only the XOR is available for future work.

Under the above conditions, our method breaks eight rounds of AUT64 within milliseconds using a standard laptop: First, we use k_G and the cipher weaknesses we identified in Section 5.1 to build a model of the Feistel function SPN

1. Determine the byte index corresponding to the first round of encryption by using the integral cryptanalytic method in Section 5.4. The first set of chosen plaintexts is

$$\mathbb{P}_1 = \left\{ \left((\texttt{0x00})^7 \parallel n \right) : n \in \{0, \ldots, 255\} \right\}$$

   Placing the byte counter n at byte index seven in the plaintext list ensures that only the first round is identified since for all cyclic permutations the seventh byte is moved in each round. Calculate $C'_{1_{r_0}} \in \mathbb{C}'_1$, where r_0 is the byte index corresponding to the output from the first round.

2. Use $C'_{1_{r_0}}$ and the corresponding counter value list to build a model of the unknown SPN part of the Feistel function F. The model is a 256 element lookup table which represents the same function as the key-defined SPN.

3. To reduce the combinatorial burden of the next step, use r_0 derived in step 1 to produce a second set of chosen plaintexts

$$\mathbb{P}_2 = \left\{ (64 - (8 \times r_0) \ll n) : n \in \{0, \ldots, 255\} \right\}$$

   Calculate $C'_{2_{r_0}}$ and $C'_{2_{r_1}}$ where r_0, r_1 are the byte indices in the ciphertext corresponding to the output from the first and second rounds respectively. Define the set of ciphertexts for this step as $\mathbb{C}_2$.

4. Use the adaptive divide-and-conquer techniques we present later in this section to reduce the remaining key space.

We now discuss our techniques for rapidly determining the remaining key parts k_τ and k_σ using the SPN model. The permutation key part k_σ can be determined efficiently by using the SPN model to calculate the expected round outputs for the next unknown round index. For example, to determine the byte index corresponding to the third round we propose the following method

1. Use the ciphertext byte lists $C'_{2_{r_0}}, C'_{2_{r_1}}$ corresponding to the first and second rounds to generate all of the possible internal cipher states $\mathbb{X}'_2$ which could be input to the third round. There will be no more than $\binom{7}{2} = 21$ possible permutations;

2. For each potential internal state $X' \in \mathbb{X}'_2$, compute the compression function G and apply its output to the SPN model. The correct permutation will yield a set of bytes which match those found in the ciphertext byte index list $C'_{2_{r_2}}$ corresponding to the output of round three.

This method of determining k_σ can be used to determine permutation elements beyond the third round by computing the possible permutations of the relevant known round index ciphertext bytes. After the first four round indices have been determined using this attack, the most efficient solution is to exhaustively search the remaining six possible keys.

For the final part of this attack, we propose a method based on the weakness identified in Section 5.2 and prior knowledge of k_σ, which efficiently determines the substitution key part. The list of output ciphertext bytes from the first round $C'_{2_{r_0}}$ contains two or more symmetric bytes. Since we have a model of the SPN and we have determined the internal permutation, we can use these symmetric bytes to determine a large number of elements in k_τ. The very worst case for an adversary is that the remaining $|k_\tau| = 2 \times 14! \approx 2^{37.3}$. On average, ten or more of the elements will be revealed and the remaining substitution key part can be exhaustively found in milliseconds. Experimentally, we found that the majority of AUT64 keys are broken within milliseconds using this method on a standard Intel i7 laptop.

7 Conclusions

In this paper, we identify significant weaknesses in the AUT64 automotive block cipher and its associated immobiliser protocol. We present a number of cryptographic AUT64 weaknesses which we combine into attacks on both eight and twenty four round implementations. Despite AUT64 having a 120-bit key, we show that in certain implementations, eight round AUT64 can be broken within milliseconds using a standard laptop, with an absolute worst case complexity of $2^{37.3}$. Also, in a concrete immobiliser system, the security of 24 round AUT64 is no more than 48.3 bits due to the use of a key derivation function that determines part of the key based on the transponder ID. We also have strong indications that the studied system makes use of manufacturer-wide keys, which reduces the attack time to a single communication with the transponder and negligible computation.

This paper contributes to the mounting evidence that it is imperative for the automotive industry to discontinue using proprietary cryptographic implementations and for them to move towards standardised algorithms and peer-reviewed protocols. We also make a novel contribution to the literature on the cryptanalysis of generalised Feistel ciphers with key-dependent permutations and S-Boxes.

References

[Atm06] Atmel. *Read/Write Crypto Transponder for Short Cycle Time*, 09 2006. Rev. 4682D-RFID-09/06.

[BBK08] Elad Barkan, Eli Biham, and Nathan Keller. Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. *Journal of Cryptology*, 21(3):392–429, Jul 2008.

[BF03] M. Bruhnke and F. Friedrich. Method of cryptological authentification in a scanning identification system, January 21 2003. US Patent 6,510,517.

[BGS+05] Stephen C. Bono, Matthew Green, Adam Stubblefield, Ari Juels, Aviel D. Rubin, and Michael Szydlo. Security Analysis of a Cryptographically-enabled RFID Device. In *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, SSYM'05, pages 1–1, Berkeley, CA, USA, 2005. USENIX Association.

[Bog08] Andrey Bogdanov. *Linear Slide Attacks on the KeeLoq Block Cipher*, pages 66–80. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[Bru10] Richard A Brualdi. *Introductory Combinatorics*, page 556. Pearson/Prentice Hall, 5th edition, 2010.

[BS91] Eli Biham and Adi Shamir. *Differential Cryptanalysis of DES-like Cryptosystems*, pages 2–21. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991.

[Com95] European Commission. Commission Directive 95/56/EC, Euratom of 8 November 1995 adapting to technical progress Council Directive 74/61/EEC relating to devices to prevent the unauthorized use of motor vehicles, 1995.

[DES77] DES. Data Encryption Standard. In *FIPS PUB 46, Federal Information Processing Standards Publication*, pages 46–2, 1977.

[DHW+12] Benedikt Driessen, Ralf Hund, Carsten Willems, Christof Paar, and Thorsten Holz. Don't trust satellite phones: A security analysis of two satphone standards. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 128–142, Washington, DC, USA, 2012. IEEE Computer Society.

[EKM+08] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T. Manzuri Shalmani. On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoq Code Hopping Scheme. In *Advances in Cryptology – CRYPTO'08*, volume 5157 of *LNCS*, pages 203–220. Springer, 2008.

[FDC10] Aurélien Francillon, Boris Danev, and Srdjan Capkun. Relay attacks on passive keyless entry and start systems in modern cars. *IACR Cryptology ePrint Archive*, 2010:332, 2010.

[GKN+08] Tim Güneysu, Timo Kasper, Martin Novotný, Christof Paar, and Andy Rupp. Cryptanalysis with COPACOBANA. *IEEE Transactions on Computers*, 57(11):1498–1513, 2008.

[GOKP16] Flavio D. Garcia, David Oswald, Timo Kasper, and Pierre Pavlidès. Lock It and Still Lose It — On the (In)Security of Automotive Remote Keyless Entry Systems. In *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX, 2016. USENIX Association.

[Gol97] Jovan Dj. Golić. *Cryptanalysis of Alleged A5 Stream Cipher*, pages 239–255. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[Ker83] Auguste Kerckhoffs. La cryptographie militaire, 1883.

[KW02] Lars Knudsen and David Wagner. *Integral Cryptanalysis*, pages 112–127. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[Mat94] Mitsuru Matsui. *Linear Cryptanalysis Method for DES Cipher*, pages 386–397. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

[NTW10] Karsten Nohl, Erik Tews, and Ralf-Philipp Weinmann. Cryptanalysis of the DECT Standard Cipher. In *Proceedings of the 17th International Conference on Fast Software Encryption*, FSE'10, pages 1–18, Berlin, Heidelberg, 2010. Springer-Verlag.

[OV16] Jan C. Ours and Ben Vollaard. The engine immobiliser: A non-starter for car thieves. *Economic Journal*, 126(593):1264 – 1291, 2016.

[PPPM13] P. Papantonakis, D. Pnevmatikatos, I. Papaefstathiou, and C. Manifavas. Fast, FPGA-based Rainbow Table creation for attacking encrypted mobile communications. In *2013 23rd International Conference on Field programmable Logic and Applications*, pages 1–6, Sept 2013.

[Riv95] Ronald L. Rivest. *The RC5 encryption algorithm*, pages 86–96. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[SIR02] Adam Stubblefield, John Ioannidis, and Aviel D. Rubin. Using the Fluhrer, Mantin, and Shamir Attack to Break WEP. In *NDSS*, 2002.

[SK96] Bruce Schneier and John Kelsey. *Unbalanced Feistel networks and block cipher design*, pages 121–144. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[Sor84] Arthur Sorkin. Lucifer, a cryptographic algorithm. *Cryptologia*, 8(1):22–42, 1984.

[SSVV14] Pouyan Sepehrdad, Petr Sušil, Serge Vaudenay, and Martin Vuagnoux. *Smashing WEP in a Passive Attack*, pages 155–178. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[Ste99] Frank Stevenson. Cryptanalysis of contents scrambling system, 1999.

[SW06] Yaniv Shaked and Avishai Wool. *Cryptanalysis of the Bluetooth E 0 Cipher Using OBDD's*, pages 187–202. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[TEM98] TEMIC Semiconductors. *Standard Read/Write Crypto Identification IC*, 10 1998. Rev. A1.

[VGB12] Roel Verdult, Flavio D. Garcia, and Josep Balasch. Gone in 360 Seconds: Hijacking with Hitag2. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 237–252, Bellevue, WA, 2012. USENIX.

[VGE15] Roel Verdult, Flavio D. Garcia, and Baris Ege. Dismantling Megamos Crypto: Wirelessly Lockpicking a Vehicle Immobilizer. In *Supplement to the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 703–718, Washington, D.C., 2015. USENIX Association.

[WTHH11] Michael Weiner, Erik Tews, Benedikt Heinz, and Johann Heyszl. *FPGA Implementation of an Improved Attack against the DECT Standard Cipher*, pages 177–188. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

7D	56	99	65	8C	74	82	83
9B	92	7B	A1	AA	B0	64	CF
B9	DE	5D	ED	C8	FC	46	0B
D7	1A	3F	29	C6	38	28	47

(a) T_D key derivation table.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	2	4	6	8	A	C	E	3	1	7	5	B	9	F	D
0	3	6	5	C	F	A	9	B	8	D	E	7	4	1	2
0	4	8	C	3	7	B	F	6	2	E	A	5	1	D	9
0	5	A	F	7	2	D	8	E	B	4	1	9	C	3	6
0	6	C	A	B	D	7	1	5	3	9	F	E	8	2	4
0	7	E	9	F	8	1	6	D	A	3	4	2	5	C	B
0	8	3	B	6	E	5	D	C	4	F	7	A	2	9	1
0	9	1	8	2	B	3	A	4	D	5	C	6	F	7	E
0	A	7	D	E	4	9	3	F	5	8	2	1	B	6	C
0	B	5	E	A	1	F	4	7	C	2	9	D	6	8	3
0	C	B	7	5	9	E	2	A	6	1	D	F	3	4	8
0	D	9	4	1	C	8	5	2	F	B	6	3	E	A	7
0	E	F	1	D	3	2	C	9	7	6	8	4	A	B	5
0	F	D	2	9	6	4	B	1	E	C	3	8	7	5	A

(b) T_{offset} compression function look up table.

T_U								T_L							
1	0	3	2	5	4	7	6	4	5	6	7	0	1	2	3
0	1	2	3	4	5	6	7	5	4	7	6	1	0	3	2
3	2	1	0	7	6	5	4	6	7	4	5	2	3	0	1
2	3	0	1	6	7	4	5	7	6	5	4	3	2	1	0
5	4	7	6	1	0	3	2	0	1	2	3	4	5	6	7
4	5	6	7	0	1	2	3	1	0	3	2	5	4	7	6
7	6	5	4	3	2	1	0	2	3	0	1	6	7	4	5
6	7	4	5	2	3	0	1	3	2	1	0	7	6	5	4
3	2	1	0	7	6	5	4	5	4	7	6	1	0	3	2
2	3	0	1	6	7	4	5	4	5	6	7	0	1	2	3
1	0	3	2	5	4	7	6	7	6	5	4	3	2	1	0
0	1	2	3	4	5	6	7	6	7	4	5	2	3	0	1
7	6	5	4	3	2	1	0	1	0	3	2	5	4	7	6
6	7	4	5	2	3	0	1	0	1	2	3	4	5	6	7
5	4	7	6	1	0	3	2	3	2	1	0	7	6	5	4
4	5	6	7	0	1	2	3	2	3	0	1	6	7	4	5
2	3	0	1	6	7	4	5	6	7	4	5	2	3	0	1
3	2	1	0	7	6	5	4	7	6	5	4	3	2	1	0
0	1	2	3	4	5	6	7	4	5	6	7	0	1	2	3
1	0	3	2	5	4	7	6	5	4	7	6	1	0	3	2
6	7	4	5	2	3	0	1	2	3	0	1	6	7	4	5
7	6	5	4	3	2	1	0	3	2	1	0	7	6	5	4
4	5	6	7	0	1	2	3	0	1	2	3	4	5	6	7
5	4	7	6	1	0	3	2	1	0	3	2	5	4	7	6

(c) T_U and T_L round key derivation tables.

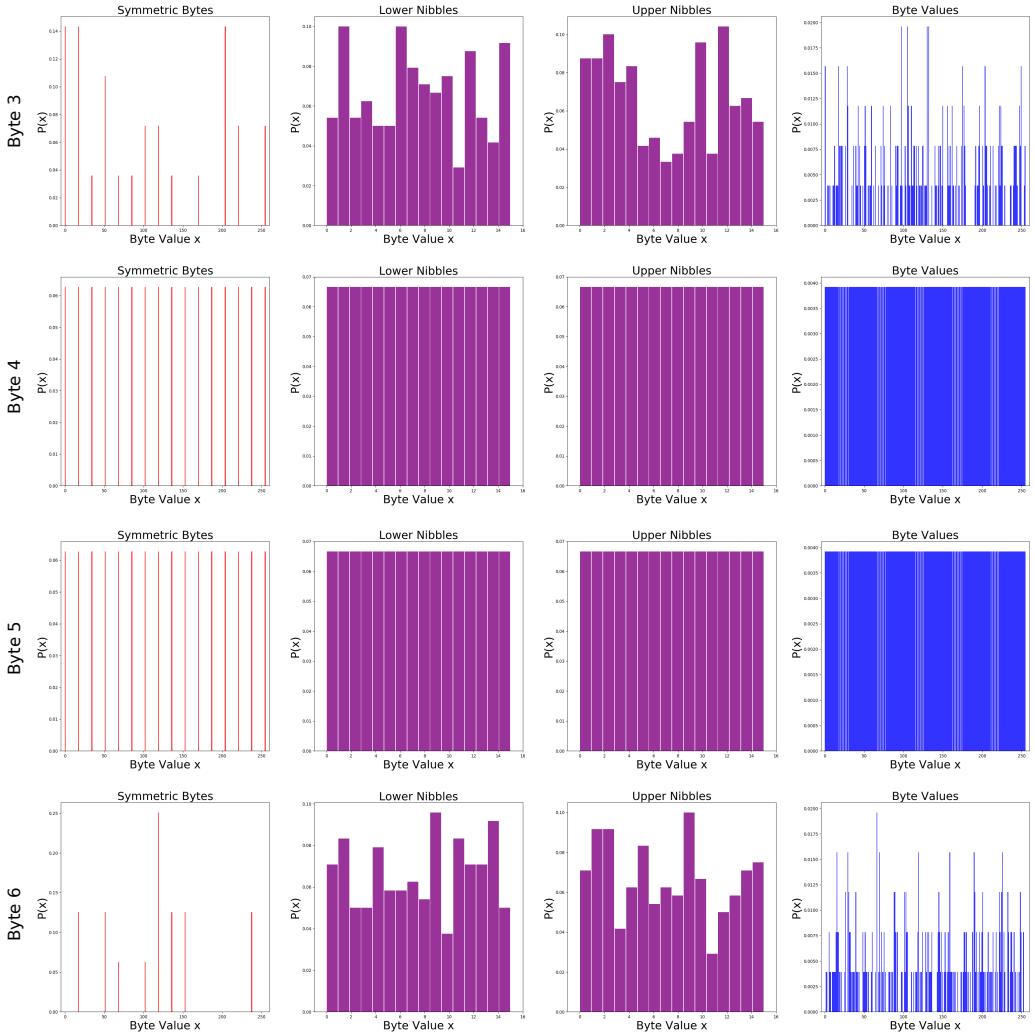


Figure 8: Histograms demonstrating the integral cryptanalytic technique which we apply in Section 5.4. Each histogram represents the cyphertext byte values which are output at a specific byte index over a set of adversarially chosen plaintexts. Ciphertext byte indices 3, 4, 5 and 6 are shown along the y-axis. The x-axis is composed from left-to-right showing the symmetric byte values, the lower nibble values, the upper nibble values and finally the complete byte values. This method of presenting the ciphertexts allows the permutation key part to be divide and conquered. In this case byte indices 4 and 5 are exposed by their uniform byte distributions as originating from the first two rounds of encryption.