A Cryptanalysis of The GOST Block Cipher

Student Name: A.I. Johnstone
Supervisor Name: K. Southern
Submitted as part of the degree of BSc Computer Science to the
Board of Examiners in the Department of Computer Sciences, Durham University

Abstract—Testing the limits of past and present encryption methods is a key area within cryptography that ensures the security of the working world. New ways of attacking ciphers are constantly being developed; however, this paper will discuss a potentially powerful yet overlooked boomerang-based method known as a sandwich attack, which was created in 2010 and has barely been used since. Although originally designed to break the KASUMI block cipher, the sandwich attack has not been utilised on more robust block ciphers, and so this paper aims to discuss the research question: Is a sandwich attack a feasible method for breaking the GOST block cipher? In order to establish whether a sandwich attack is an efficient method to use against GOST, a ground-up style approach is taken in this paper, starting from a replication of a sandwich attack on KASUMI and a related-key boomerang attack on GOST, which are both then used as a platform to build a sandwich attack on GOST. Although the attack on KASUMI could be successfully replicated, the results for the related-key boomerang attack on GOST could not be. However, with a slight adjustment to the differentials, comparable results could be achieved. Utilising both of these implementations, it was established that a sandwich attack on GOST would in fact be a feasibly efficient method, as the sandwich distinguisher was found to have a probability at least as good as the related-key boomerang distinguisher. Although this has not been tested experimentally, the results acquired from theory testing show good promise and suggest that a sandwich attack on GOST would be no more complex than the standard boomerang attack.

Index Terms—Code Breaking, Data Encryption, GOST, Related-Key Attack, Sandwich Attack, Standards

1 Introduction

Advances in modern technology have meant that there is a constant need for encryption within our everyday lives. Subsequently, these encryption methods have also needed to become more and more secure as attacks from hostile users have become increasingly advanced. As such, the field of cryptography is always trying to find new ways to ensure security or conversely, find new ways to breach this security so that they can stay ahead of malicious adversaries.

This work hopes to extend the studies surrounding the generation of best-known attacks, specifically looking at block cipher encryption. We achieve this by utilising several existing techniques and ciphers to create a platform from which we create a new attack on the GOST block cipher. We then attempt to show that this new method is at least as good as existing methods, with the potential to be a new best-known attack on GOST.

The work conducted in this project could be of significant importance to the cryptographic community, as we show that the sandwich attack on KASUMI as presented in [1] can be generalised for other block ciphers. Specifically, we present a theoretical analysis of a sandwich attack on the GOST block cipher and show that it can provide greater probabilities than a similar style of attack - the related-key boomerang attack. Due to the nature of the sandwich attack being built upon a related-key boomerang attack, we are able to suggest that in ciphers where a related-key boomerang attack is applicable, a sandwich attack could be used instead and would likely obtain better results. The way in which this would impact the wider cryptographic community is because the current widely used encryption standard, AES, is susceptible to related-key boomerang attacks (as presented in [2], [3] and [4] to provide a few examples). Since in this project we are able to present an attack that works similarly to the related-key boomerang attack, but that is more effective, it would be reasonable to say that with some further generalisation of the sandwich attack, it could be applied to AES to achieve a new best-known attack. Although it is unlikely that the sandwich attack would be able to break AES to such an extent that a new method of encryption would need to be devised, it could present a weakness in the cryptosystem that allows for more of AES to be broken in reasonable time.

1.1 Deliverables

The research question asked in this paper is: 'Is a sandwich attack a feasible method for breaking the GOST block cipher?'

To try and answer this question, a number of project deliverables were established and categorised into basic, intermediate and advanced deliverables.

The basic deliverables were to replicate results from existing work, notably, to replicate the results of a sandwich attack on KASUMI from [1] and to replicate a related-key boomerang attack on GOST from [5]. The purpose of replicating the results from [1] was to establish that the sandwich attack was, in fact, a correct method and could produce repeatable results. We were able to reproduce the results from [1] by utilising the basic pseudocode provided, which was then translated into Python alongside using a modified version of the KASUMI implementation from [6]. The purpose of replicating the results from [5] was to gain a better understanding of the GOST block cipher

and also to look at how the related-key boomerang attack worked so as to better understand how to implement a sandwich attack on GOST. Also, the attack presented in [5] is purely theoretical, and there is no experimental verification of this attack. Therefore, we present the first experimental verification of this attack within this project. We were unable to directly achieve the same expected results as in [5] as we experienced some issues regarding the provided differentials, despite the provided differentials being the same as those used in other papers (e.g. [7]). However, we did manage to get comparable results after adjusting the differentials, translating the pseudocode from [5] to Python and utilising an adapted version of GOST from [8]. The reason for replicating the results from two papers was to allow us to gain a deeper understanding of how all of the key components needed to answer the research question worked and how they could later be combined.

The intermediate deliverable was to establish whether a sandwich attack on GOST would be feasible in theory based on its computational complexity. After establishing where to put the 'filling layer', we were able to work out that there was no additional work required to create the sandwich distinguisher compared to the related-key boomerang distinguisher. This means that, in theory, the sandwich attack on GOST should be no more computationally complex than a related-key boomerang attack. This led us to conclude that the sandwich attack could be a feasible method for breaking the GOST block cipher and subsequently answering the research question.

In the initial plan for this project, two advanced deliverables were presented however, they were dependent on whether the intermediate deliverable was successful. The first advanced deliverable was for if we were able to show theoretically that the sandwich attack was somewhat feasible. From this, we would then go on to implement the attack to see if we could prove our results experimentally. The second advanced deliverable was for the scenario where a sandwich attack on GOST proved to be infeasible and so rather than attempting to implement the attack (as this would be pointless given that it is more complex than the existing related-key boomerang attack) we would instead add to the existing documentation of the sandwich attack providing more detailed descriptions and pseudocode for it. However, due to issues in the project regarding incomplete and outdated resources as well as implementation issues, the project fell behind schedule, and subsequently, we were unable to attempt any of the advanced deliverables.

2 Related Work

In this section, a commentary on existing papers surrounding the subject area of this project will be presented, initially discussing and defining some baseline terms that are then expanded upon and developed into techniques that have been used in this project.

2.1 Term definition

To be able to gain an understanding of some of the more complex techniques used in this project, a selection of useful terms has been provided and explained below.

2.1.1 Feistel Cipher

When constructing a block cipher, it is common for the 'main' encryption part of the machine to be created using a Feistel cipher [9, pp. 11-14]. Named and designed by Horst Feistel, who did pioneering research whilst working for IBM, a Feistel cipher (also called a Feistel network) is a symmetric key algorithm. A Feistel cipher works by taking two inputs, a plaintext and a key, and performing multiple 'rounds' using these inputs to produce a ciphertext output. More formally, a Feistel function works as follows:

Let the inputs to the Feistel function be P and K where $P = R_0||L_0$ is the plaintext input comprised of two equal halves R_0 and L_0 and where K is the key. The ciphertext outputs L_i and R_i are obtained by performing a combination of XORs, switches and applying the round function. The round function f_i is a pseudorandom function that uses the key K to add non-linearity to the Feistel network and ensure that the encryption cannot be undone trivially.

Figure 1 showcases the process of performing the round function on a Feistel network.

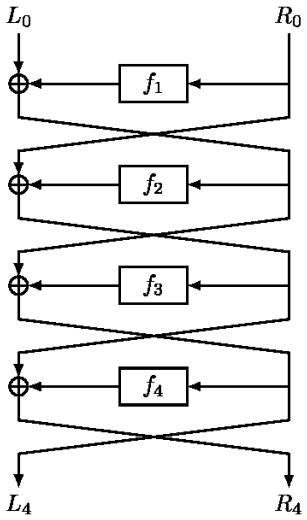

Fig. 1. A diagram showing the schema of a 4 round Feistel cipher [10]

2.1.2 S-Boxes

Used in block ciphers to help obey Shannon's property of confusion [11, pp. 51-82], substitution boxes (S-boxes) are mathematically non-linear vectorial Boolean functions [12, pp. 398-470]. However, in general, an S-box takes some number of input bits, m, and transforms them into some number of output bits, n. In this instance, m doesn't have to be equal to n, for example, DES takes a 6-bit input and produces a 4-bit output [13, 243-250]. S-boxes are used to help obscure the relationship between the plaintext and the ciphertext and to provide nonlinearity so that a block cipher cannot be trivially decrypted.

2.2 Block Ciphers

A key part of this project involves analysing block ciphers. Block ciphers are a type of private key cryptosystem that operate on fixed-length groups of bits, known as blocks [14, pp. 22]. These block ciphers take a binary input string

and then use an operation, for example, a Feistel cipher, to encrypt this input using a chosen second input called a key. In cryptography, these inputs are known as plaintexts, and the outputs are called ciphertexts. Within this paper, two primary block ciphers are looked at; however, for ease of understanding, this section mentions three.

2.2.1 DES

The Data Encryption Standard (DES) (First released in FIPS-46) is a block cipher developed in the early 1970s by IBM that utilises a balanced Feistel network of 16 rounds to encrypt 64-bit data blocks using a 56-bit key [15]. Although originally derived from the block cipher Lucifer, developed by Horst Feistel, DES was modified by the NSA (National Security Agency) to be more resistant to differential attacks before it was accepted as a national encryption standard in 1976 [13]. Despite being more resistant to differential cryptanalysis, DES was also made to be weaker against brute force attacks by the NSA, as they wanted to be able to break DES if it ever became necessary. Although upon release, DES was known not to be completely secure, it has since been proven that it is no longer a viable method of encryption. Notably in 1998 the EFF DES cracker (aka Deep Crack) [16] could brute force a DES key in a matter of days. Hence, in 2005, it was withdrawn from the encryption standard and replaced with a new block cipher called AES (Advanced Encryption Standard) [17]. Despite this, some alternate versions of DES exist that are still considered secure, such as TripleDES [18]. Although not used as an encryption standard any more, DES is still an area of interest for researchers as it can give them a reliable benchmark for new attacks or potentially provide them with information about how AES will react to an attack. Although this project does not directly use DES, it can still be useful to use as a comparative benchmark for the proposed attacks, as significantly more research has been done on DES compared to KASUMI or GOST.

2.2.2 GOST

GOST is a Soviet and Russian government standard symmetric key block cipher developed in 1989 (originally called Magma) [19] but that has since had revisions and is still used today (now called Kuznyechik) [20]. GOST (an acronym derived from gosudarstvennyy standard, which translates to government standard) is similar to DES in that it is a 64-bit block size and utilises a Feistel network; however, some of the main differences between the two ciphers are listed below:

- GOST uses a key size of 256 bits
- GOST uses 32 rounds of the Feistel function
- The S-boxes of GOST take a 4-bit input and produce a 4-bit output
- The S-boxes of GOST can be changed

Despite these differing factors appearing to make GOST a more secure cipher compared to DES, it was found in 2011 that GOST could be broken quite easily and was even called a 'deeply flawed cipher' by Nicolas Courtois [21].

More formally, GOST is a Feistel-based block cipher where its round function is defined as follows: Let the 64-bit plaintext block (the input) be denoted as:

$$X = L_0||R_0$$

where $L_0, R_0 \in \{0,1\}^{32}$ are the left and right 32-bit halves, respectively. And let the 256-bit master key be represented as eight 32-bit words:

$$K = (k_1, k_2, ..., k_8), k_j \in \{0,1\}^{32}$$

The subkeys for rounds $1-24$ cycle through $(k_1, ..., k_8)$ in order, and rounds $25-32$ cycle through in reverse.

The core of each Feistel round is the function $G_{K_i} : \mathbb{Z}_{2^{32}} \to \mathbb{Z}_{2^{32}}$, defined by:

$$G_{K_i}(u) = ROL_{11}(S((u + K_i) \mod 2^{32})$$

where: $(u + K_i) \mod 2^{32}$ denotes addition modulo 2^{32}, ensuring the result remains a 32-bit word, S represents the S-Box transformation function and ROL_{11} is the left circular rotation by 11 bit positions on a 32-bit word.

Each round transforms the state $L_{i-1}||R_{i-1}$ to $L_i||R_i$ as follows:

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus G_{K_i}(R_{i-1})$$

After 32 rounds, the output of the cipher is the pair (L_{32}, R_{32}), which is then swapped and combined to produce the 64-bit ciphertext: $C = R_{32}||L_{32}$. Diagrams showing the workings of this formal definition for the GOST round function can be seen in Figure 16 (which shows one round) and Figure 17 (which shows 32 rounds) as part of Appendix B.

Although GOST has been considered broken, the attack that would perform this is infeasible in practice. This is why GOST has been chosen for this project, as we would like to see if a sandwich attack can either further break GOST or provide a more feasible attack on it. Alongside choosing GOST, we are also going to choose the S-boxes defined in the original publication of GOST [22] as this should provide the best comparison to the first 'retro' version of GOST (aka Magma).

2.2.3 KASUMI

KASUMI is a Feistel network block cipher developed for use in mobile communications for 3GPP in 1995 (3rd Generation Partnership Project) [23] and is based on the MISTY1 cipher [24]. KASUMI uses 128-bit keys, 64-bit blocks and generally uses 8 rounds of its Feistel function. Although [25] states that KASUMI was developed to be stronger than MISTY1, [1] showed that it was actually weaker due to the modifications made for use with 3GPP.

2.3 Types of Attack

This next section will cover the different styles of attack that exist and which ones have been utilised within this project.

2.3.1 Ciphertext Only attack

A ciphertext-only attack (COA), also called a known ciphertext attack, is the most basic form of attack where an attacker only has access to intercepted ciphertexts [26, pp. 8-9]. This type of attack is commonly seen in practice as it is generally much easier to obtain just the ciphertext compared to a plaintext-ciphertext pair or have knowledge of the

encryption system used. A COA is considered successful if the attacker is able to obtain any previously unknown information, deduce the corresponding plaintexts to their ciphertexts or deduce the encryption key.

2.3.2 Known Plaintext Attack

A known plaintext attack (KPA) is a type of attack method originally developed during World War two at Bletchley Park under the term 'crib' [27]. A KPA is an extension of a COA, as the attacker now has access to plaintext-ciphertext pairs. For a KPA, an attacker tries to exploit weaknesses within the encryption system by utilising techniques such as frequency analysis or pattern matching. The more pairs the attacker has access to, the easier it is to work out the structure of the encryption method and hence making the attack easier. In this style of attack, only information that is intercepted can be used, as it is assumed that the attacker has no knowledge of the encryption system used by the sender [26].

Figure 2 shows how an attacker would obtain the plaintext-ciphertext pairs necessary to perform a KPA.

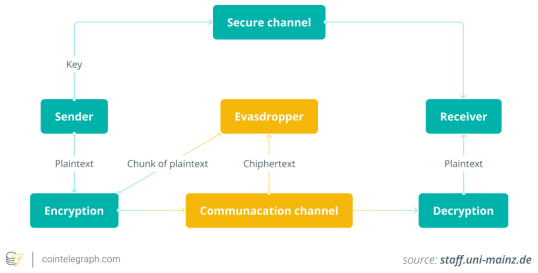

Fig. 2. A visual representation of how an attacker can acquire pairs of plaintexts and ciphertexts in a KPA [28]

2.3.3 Chosen Plaintext Attack

Unlike with COAs and KPAs, a chosen plaintext attack (CPA) doesn't rely on an attacker being able to intercept messages. Instead, a CPA assumes that the attacker has access to an encryption oracle (that is viewed as a black box) and is able to obtain ciphertexts for arbitrary plaintexts [26]. The aim of this attack is for the attacker to obtain all parts of the encryption key.

There are two variants of a CPA: batch CPA and adaptive CPA. Batch CPA (often just called CPA) is where an attacker chooses a number of plaintexts before seeing any corresponding ciphertexts. All of the plaintexts are then sent to the oracle, and all the ciphertexts are returned (in such a way that it creates pairs of plaintexts and ciphertexts). Contrastingly, in an adaptive CPA (most often referred to as CPA2), an attacker can request the ciphertexts of additional plaintexts after having seen any number of plaintext-ciphertext pairs.

CPAs are generally considered more powerful than COAs and KPAs, as the attacker is able to target specific terms or phrases without having to wait for them to occur naturally and be intercepted. This means that any cipher considered secure against a CPA can also be considered secure against COAs and KPAs. This is because an attacker can simulate a less powerful attack by ignoring some of the information they have. For example, an attacker using a CPA can simulate a KPA by ignoring their ability to query new plaintexts (using the oracle to gain new plaintext-ciphertext pairs) and instead just utilising a set of plaintexts that already exist. Equally, an attacker using a KPA can simulate a COA by ignoring the plaintexts that are known to them. This creates a hierarchy of attack types:

$$\text{CPA} > \text{KPA} > \text{COA}$$

where the implication is that if a cryptosystem is secure against a particular attack, then it is also secure against attacks weaker (lower in the hierarchy) than it.

2.3.4 Chosen Ciphertext Attack

Building upon a CPA, a chosen ciphertext attack (CCA) now grants an attacker a decryption oracle (also viewed as a black box) as well as an encryption oracle and so is able to obtain plaintexts from arbitrary ciphertexts as well as ciphertexts from arbitrary plaintexts. This now allows an attacker to defeat schemes that were considered secure under CPA, for example, the El Gamal cryptosystem is semantically secure under CPA but can be trivially defeated by CCA.

Like CPAs, there are a few variants of CCAs, namely, non-adaptive, adaptive and lunchtime. The non-adaptive (or batch) and adaptive variants work similarly to those for CPA. Where an attacker chooses the ciphertexts to be decrypted before seeing any plaintexts is non-adaptive CCA (generally just called CCA), and where an attacker may choose additional ciphertexts to decrypt after seeing some number of plaintexts is adaptive CPA (often abbreviated to CCA2). The lunchtime variant, however, works a bit differently and is somewhat of a combination of the two previous methods. The lunchtime variant (shortened to CCA1) allows an attacker to make adaptive chosen-ciphertext queries (as in CCA2) but only up until a certain point. After this point, the attacker can no longer make queries to the oracle and may only use the information already gathered to continue the attack. The idea behind the lunchtime variant stems from the idea that an attacker may be able to access a user's computer whilst they are on lunch break, but as soon as the user is back, they no longer have access, so as to avoid suspicion.

In terms of the previously mentioned hierarchy of attack types, CCA is considered stronger than CPA, and so the hierarchy now looks as follows:

$$\text{CCA} > \text{CPA} > \text{KPA} > \text{COA}$$

2.3.5 Related Key Attacks

First proposed by Biham in [29] a related key attack (RKA) is a form of cryptanalysis where an attacker can see how a cipher works under a set of initially unknown keys but where a mathematical relationship connecting the keys is known [14, pp. 108]. This mathematical relationship allows the attacker to exploit certain parts of a cipher so as to obtain results that are statistically more probable to occur than random.

2.3.6 Distinguishing Attacks

A distinguishing attack is a form of algorithm that tries to show that a cryptosystem exhibits non-random behaviour

[30, pp. 358]. Specifically, a distinguishing attack tries to break the indistinguishability property of a cryptosystem by showing that an adversary can correctly identify the corresponding plaintexts to a set of ciphertexts with probability significantly better than random. In reference to the security of a cryptosystem, it is generally considered a basic requirement to be secure for indistinguishability under CPA (often shortened to IND-CPA). What this means is that if an attacker is unable to break the indistinguishability property using a CPA, then the cryptosystem is considered IND-CPA secure. IND-CPA is equivalent to semantic security [30, pp. 1176-1177] and the terms are often used interchangeably. Indistinguishability can also be applied to CCAs, namely: indistinguishability under CCA1 (IND-CCA1) and indistinguishability under CCA2 (IND-CCA2). Security for either of the definitions of indistinguishability under CCA implies security under the previous level, i.e. if a scheme is IND-CCA2 secure, it is also IND-CCA1 and IND-CPA secure, but if a scheme is only IND-CCA1 secure, then it is not IND-CCA2 secure but is IND-CPA secure.

All of the attacks that were performed as part of this project attempt to break the indistinguishability property under adaptive chosen ciphertext attacks utilising related keys, or in short, we are trying to break IND-RK-CCA2. This is because both encryption and decryption oracles were used within the related-key structure in order to establish a probability greater than random for identifying plaintext-ciphertext pairs.

2.4 Attack Methods

The following section will provide a very basic overview of some of the attacks and their related predecessors that have been used within this project.

2.4.1 Differential Attacks

In its broadest sense, differential attacks (or differential cryptanalysis) are the study of how differences in an input can affect the resulting differences in the output. When talking about block ciphers specifically, it refers to the techniques used to trace differences through the cipher to try and discover where the cipher exhibits non-random behaviour and then exploiting those properties to retrieve the secret key [31]. The discovery of Differential cryptanalysis is generally attributed to Biham and Shamir, where, in [31], they proposed an attack on DES in the late 1980s. However, it was later revealed by Coppersmith that IBM had known about differential cryptanalysis since 1974 [13].

Differential cryptanalysis is usually a CPA, meaning that an attacker must be able to obtain some ciphertext output from a set of plaintexts they have chosen. The basic version of the attack relies on pairs of plaintexts related by a constant difference (usually XOR). The attacker then computes corresponding pairs of ciphertexts and their differences, hoping to discover patterns in their distributions that lead to the cipher being distinguished from random. The resulting pair of differences is called a differential. However, in the basic version of the key recovery attack, the attacker is instead trying to ascertain the secret key used in the cipher. Since its (re)discovery, differential cryptanalysis has been the building block for most modern differential-based techniques.

2.4.2 Boomerang Attack

One such attack that builds upon the differential attack is known as a boomerang attack. First proposed in [32] by Wagner in 1999, the boomerang attack is a form of CCA that splits the cipher into two consecutive stages, allowing for a differential that doesn't have to cover the entire cipher and instead only needs to cover part of it. During the attack, a so-called 'quartet' structure is sought after, where this quartet contains four plaintexts that adhere to certain restrictions. To obtain these quartets, an attacker must first choose a plaintext, P_a, then XOR this with some chosen differential α to get a new plaintext P_b. After encrypting these plaintexts with the algorithm in question to obtain ciphertexts C_a and C_b, these ciphertexts are then XOR'ed with a known differential δ to obtain two new ciphertexts C_c and C_d. Ciphertexts C_c and C_d are then decrypted through the algorithm to acquire the remaining two plaintexts P_c and P_d. If all four plaintexts fit the quartet conditions, then they are said to be a right quartet.

Since its release, the boomerang attack has seen lots of use and in some cases produces very good results for attacks on block ciphers as seen in [33], [34], [35]. Although most of these attacks are variants of the original CCA boomerang attack, such as rectangle attacks or amplified boomerang attacks, they are mostly RK-CCA. The way in which the related-key boomerang attack differs from the regular version refers to the keys under which parts of the attack are performed. In the regular boomerang attack, both the first and second sub-ciphers operate under a key K. However, in the related-key boomerang attack, the first sub-cipher operates on key K and the second sub-cipher operates on key $K' = K \oplus \Delta$ where Δ is the chosen key difference (some mathematical relationship between the two keys).

With related-key boomerang attacks being prevalent in the field of cryptography, it means that there have been a number of related-key boomerang attacks performed on GOST, notably [33], [36], [5], [37]. For this project, we chose [5] as a baseline related-key boomerang attack on GOST.

2.4.3 Sandwich Attack

Although initially presented in 2010 by Dunkelman et al. [38], the Sandwich Attack has seen very little usage over the years, with the only other paper being by Jana et al. in 2019 [39]. As such, these papers are the only source of information on how the attack works. However, the sandwich attack is a follow-on from a related-key boomerang attack, so some of the workings of the attack can be ascertained from that. In [38], they initially describe the sandwich attack by first explaining a related-key boomerang attack and then advancing into the sandwich attack. As mentioned before, in a related-key boomerang attack, the cipher is split into two consecutive sections; however, in a sandwich attack, the cipher is now split into three cascading sub-ciphers. A sandwich attack still utilises the sub-ciphers E0 and E1; however, one of them will be one round shorter compared to the boomerang attack. This is due to the new sub-cipher, M (sometimes referred to as the filling and hence the name sandwich attack), operating on exactly one round. The round that M operates on can be taken from either E0 or E1 and is chosen based on which provides the highest

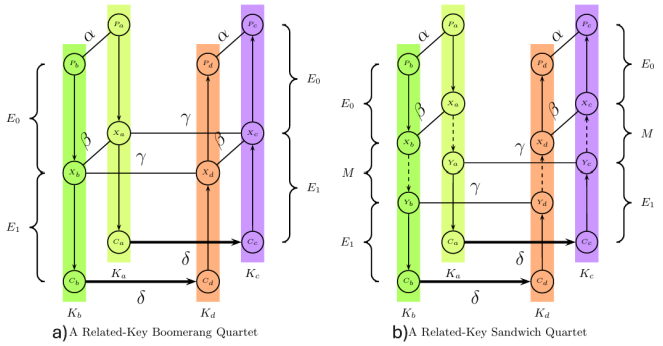

Fig. 3. Two diagrams showing the quartet structure of a related-key boomerang attack (a - left) and a sandwich attack (b - right) [38]

probability of a specific differential appearing. The purpose of this extra round is to increase the probability of a right quartet occurring and therefore reducing the number of plaintexts needed for the attack. Specifically, in [38] the sandwich attack works on KASUMI because the filling layer (round 5) was chosen so that there is no difference in the input and output differentials for that round, i.e. a simple Feistel switch is performed with no other changes occurring.

Figure 3 shows the basic quartet structure for both a related-key boomerang and a sandwich attack. From diagram b, it is clear to see where the extra middle stage has been added to the process and subsequently how it differs from the related key boomerang attack in diagram a.

3 Methodology

This project aims to discuss the possibility of a more generalised sandwich attack, specifically looking at its feasibility when used against the GOST block cipher. To aid in trying to accomplish this, we have extended upon ideas discussed in the related work section by replicating the works from [38] and [5] to provide a platform from which a sandwich attack on GOST can be built.

3.1 A Sandwich Attack On KASUMI

The first stage of this project was to implement the KASUMI block cipher to be able to encrypt and decrypt arbitrary data. This would be used as a black box conforming to the operations of an attack in the style of CCA2. Using this black box, we would then go on to implement the sandwich attack presented in [38], trying to achieve similar results and prove that the sandwich attack is a viable method of cryptanalysis.

3.1.1 The Implementation of KASUMI

Before implementing KASUMI, it was first key to understand how it works. Utilising a 128-bit key and a 64-bit input/output, KASUMI is a relatively simple block cipher containing at its core an 8-round Feistel network. It starts by splitting the 64-bit input word into two 32-bit halves, Left and Right, with the right half containing the most significant bits and the left containing the least significant bits.

$$Input = R_0||L_0$$

In each round, Right is XOR'ed with the output from the round function, and then the halves are swapped.

$$L_i = F_i(KL_i, KO_i, KI_i, L_{i-1}) \oplus R_{i-1}$$
$$R_i = L_{i-1}$$

where KL_i, KO_i, KI_i are round keys for the i^{th} round.

Depending on whether the round in question is even or odd, there is a slightly different round function that is applied. In either case, each round is comprised of a combination of two functions FL_i and FO_i, where FO_i contains a function $FI()$.

For an odd round:

$$F_i(K_i, L_{i-1}) = FO(KO_i, KI_i, FL(KL_I, L_{i-1}))$$

For an even round:

$$F_i(K_i, L_{i-1}) = FL(KL_i, FO(KO_I, KI_i, L_{i-1}))$$

What this essentially means is that for odd rounds the data is passed through $FL()$ then $FO()$ whilst for even rounds it is passed through $FO()$ then $FL()$.

The output of each round function is a concatenation of the outputs of the previous round.

For the FL function, an input of a 32-bit data string, I, and a 32-bit subkey, KL_i. The subkey is split into two subkeys such that:

$$KL_i = KL_{i,1}||KL_{i,2}$$

and the input is split into two halves L, R where $I = L||R$.

The output to the FL function is the 32-bit value $(L'||R')$ where:

$$R' = R \oplus ROL(L \wedge KL_{i,1})$$
$$L' = L \oplus ROL(R \vee KL_{i,2})$$

and where $ROL()$ is a 16-bit rotate left.

The $FO()$ function takes as input a 32-bit data string, I, and two 48-bit sets of subkeys KO_i and KI_i. The data input is split into two halves, $I = L_0||R_0$ and the 48-bit subkeys are subdivided into three 16-bit subkeys defined as:

$$KO_i = KO_{i,1}||KO_{i,2}||KO_{i,3}$$
$$KI_i = KI_{i,1}||KI_{i,2}||KI_{i,3}$$

Then for each integer $1 \leq j \leq 3$ it is defined as:

$$R_j = FI(L_{j-1} \oplus KO_{i,j}, KI_{i,j}) \otimes R_{j-1}$$
$$L_j = R_{j-1}$$

The output of the $FO()$ function is the 32-bit value $(L_3||R_3$.

The $FI()$ function used within the $FO()$ function is used to pass the 16-bit input data through two S-boxes, S7 and S9. Unlike the other functions, the $FI()$ function splits the input data and key unequally into a 9-bit and a 7-bit component. After performing some operations so as to provide non-linearity to the encryption process, the $FI()$ function then returns a 16-bit value.

These functions are then combined to create the round function, which is performed eight times before returning a 64-bit output string. The diagrams showcasing how the KASUMI block cipher works can be found in Figures 12, 13, 14 and 15 of Appendix A.

With an understanding of how KASUMI worked, we then set out to create our implementation of it, which we could manipulate for use with the sandwich attack. Although the code provided in [23] is written in the C

programming language, it was decided during the planning phase of this project that we wanted to program using Python, as this was a more familiar language. Subsequently, the plan was to translate the provided C code into Python so that we could have a Python implementation of KASUMI, which would hopefully be easier to use and manipulate when it came to implementing the sandwich attack.

Despite the initial development process appearing to go well, we soon ran into issues with the KASUMI implementation. Specifically, we experienced issues with it not being able to provide correct encryptions. After spending a reasonable amount of time unsuccessfully trying to debug the implementation of KASUMI, it was decided that, in the interest of trying to maintain on schedule, a pre-existing implementation of KASUMI would be used. Although this would likely make the modifications needed to construct the sandwich attack harder to implement, we decided that a known-to-be-working KASUMI implementation would be more beneficial and would also save a significant portion of time we would have otherwise had to spend on debugging our original attempt to implement the cipher. The working implementation we chose to incorporate into our project came from [6], and we experienced no issues with this when testing encryption or decryption functions.

3.1.2 The Implementation of a Sandwich Attack on KASUMI

With the implementation of KASUMI working, we were then able to progress onto building the sandwich attack. The initial step was to modify the KASUMI code so that it would be able to perform as a black box and provide all the information necessary to undertake a sandwich attack. Although we speculated that this may prove difficult due to not using an implementation of KASUMI that we had written, it proved to be relatively simple to incorporate the extra functions. Mainly, this section consisted of manipulating KASUMI so that it only operated on seven rounds rather than eight, as the attack presented in [38] proposes a distinguisher for 7-round KASUMI and then utilises brute force methods for the final round. We also needed to add in new function calls to provide us with the intermediate values from the sub-cipher M, see Figure 3.

Next, we moved on to implementing the quartet conditions, which were the bulk of the sandwich attack implementation. Utilising the pseudocode provided in [38], we were quickly able to add the conditions into the project and begin testing. However, once testing began, we soon realised that none of the conditions were being met despite [38] suggesting that we should be seeing a right quartet, all three conditions being met, with probability $1/4$. After spending a long time trying to debug the code, we were unable to find any notable differences between our work and the pseudocode presented in [38]. This was suggesting that the work presented in [38] may not have been correct and was not replicable. However, we were later able to establish that the paper provided on the official International Association for Cryptologic Research (IACR) website was an outdated file containing a preprint paper, which had not been moderated and so subsequently contained a number of errors. This meant that all of the implemented code for the sandwich attack at the time was wrong and would explain why we were experiencing so many issues trying to find right quartets. Due to these errors found in the paper we had access to, we then had to search for the most up-to-date version of the paper, which proved quite difficult. However, we were eventually able to find a corrected paper, [1], which contained the correct form for the quartet conditions. From this new paper, we were then able to successfully implement the sandwich attack on KASUMI, although due to the issues surrounding the paper, this took far longer than expected and put the project quite far behind schedule.

3.2 A Related-Key Boomerang Attack On GOST

The next stage of this project was to go on to replicate the related-key boomerang attack presented in [5] in a similar way to how we implemented the sandwich attack on KASUMI. That is, creating a black box implementation of GOST to use in a related-key CCA2 style boomerang attack. The reasoning behind implementing a related-key boomerang attack on GOST is that we believe it will provide a useful midpoint between the previously implemented sandwich attack on KASUMI and the proposed sandwich attack on GOST. This mid-point can then be used as a stepping stone to help guide the project towards answering the research question. Although we say that we are attempting to replicate the results from [5], this paper only presents a theoretical attack on GOST and does not experimentally verify any of the values they predict. Hence, in this project we are providing the first such implementation for the experimental verification of the attach presented in [5] and are trying to replicate the presented theoretical results.

It is important to note at this stage that the variant of the related-key boomerang attack on GOST that is implemented as part of this project originally comes from [40]. However, we have been unsuccessful in being able to locate a readable copy of this paper, and so we utilise [5] instead, as this paper replicates the work from [40] and provides pseudocode useful for the project. Furthering on from this, it is necessary to point out that although [5] replicates the work from [40], it does then go on to say that the theory behind this attack is wrong. Specifically, [5] says that the attack presented in [40] is no better than exhaustive search. However, [5] does not go on to prove this claim experimentally. Hence, the specific related-key boomerang attack that we utilise in this paper has contrasting views as to its validity, but has no experimental data to back up either side of the claims. Furthermore, because we are creating the first experimental verification of this attack we should expect to encounter issues and potentially have to go as far to say that both the claims made originally in [40] and the claims made against [40] in [5] are incorrect.

Also, as part of this project we do not go on to implement any of the advanced techniques presented in [5] as we felt it was only necessary to use a 'basic' related-key boomerang attack on GOST given it is only being used as a stepping stone towards an implementation of a sandwich attack on GOST.

3.2.1 The Implementation of GOST

Although a more complex block cipher than KASUMI due to its being a 32-round balanced Feistel network structure,

the functionality of GOST is much simpler, specifically when talking about the round function. Figure 4 shows a diagram for how the GOST round function works, which is as follows: Let L and R be the 32-bit halves of the input to a round (GOST takes as input a 64-bit data word and a 256-bit key). A 32-bit round key modulo 2^{32} is then added to R, $R = R + K_i$. This new value then has eight 4-bit S-boxes applied to it, and the result after this is then left-shifted by eleven bits. Finally, this value is then XOR'ed with L to produce a new left half. As with most Feistel network structures, the results from the round function (the newly calculated value for L and the unchanged R) are swapped. A diagram showing a singular round and the full operation of GOST can be found in Figures 16 and 17 of Appendix B. Another key detail of GOST is that it operates in two components: the first 24 rounds and the last 8 rounds. The only difference between these sections is that the ordering of the subkeys is reversed for the final 8 rounds.

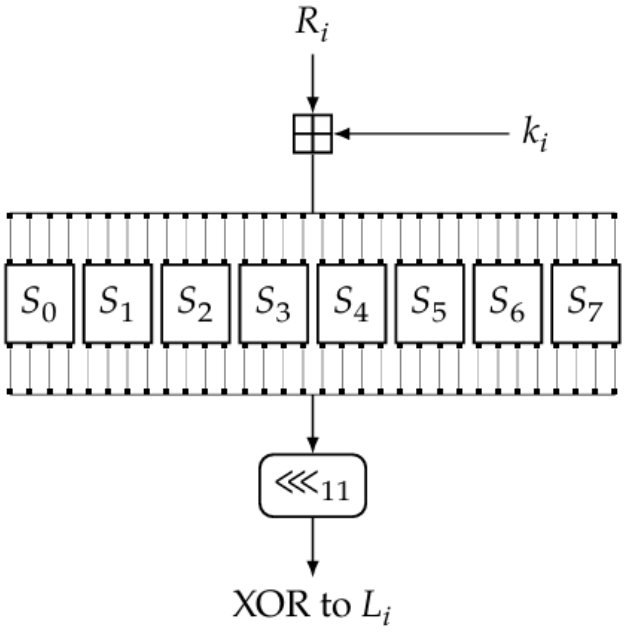

Fig. 4. A diagram showing how the GOST round function works [14, pp. 141]

Although the initial plan was to implement an original version of GOST written in Python, due to the setbacks experienced whilst developing the new Python implementation of KASUMI, we decided it was best to use a pre-made solution so as to prevent any further setbacks and help to get the project back on schedule. The existing solution we used came from [8], and we experienced no issues surrounding the incorporation of this into the project. In order to ensure that the implementation of GOST from [8] was correct, we performed a number of known answer tests (KATs) on the implementation. This was to ensure that the correct version of GOST was being used within the project and to try and help prevent issues later down the line. As part of the KATs, we tested the implementation of GOST against known plaintext-ciphertext pairs so as to ensure the validity of both the encryption and decryption functions. We also performed tests that used different S-boxes to ensure that the subkey schedule was correct.

It should be noted here that, unlike KASUMI, which uses a fixed set of S-boxes, GOST is able to use a variety of S-boxes of varying strengths. It is said that this was included as a feature by the Russian Government (who have used variations of GOST as their encryption standard since the 1970's) so that they could provide 'weaker' S-boxes to Russian companies and corporations who they wished to spy on whilst giving the stronger S-boxes to those that held sensitive information that they did not want to be leaked. As such, for this project we will be utilising the S-boxes defined in [41, pp. 331-335] as these are generally considered the 'standard' S-boxes.

3.2.2 The Implementation of a Related-Key Boomerang Attack On GOST

After successfully implementing the existing GOST code, we then moved on to incorporating the related-key boomerang attack from [5] into the project. The first stage was to manipulate the GOST implementation into a black box suitable for the related-key boomerang attack. As with modifying KASUMI into a black box, we expected this to be troublesome as the GOST implementation was not code we had written and so might not be best suited to being moulded the way we needed. Although it was more difficult to add the extra functions compared to KASUMI, this process was still easier than we had anticipated. This section mainly focussed on creating a function to return the value after the first 24 rounds but before the last 8 rounds of the cipher, as well as ensuring the cipher code worked in the same format as the attack code.

Continuing, we then added the quartet conditions to the attack code, which, like the sandwich attack, is a large proportion of the code needed for a related-key boomerang attack. After including these, we then set about testing the code to see if we could obtain any right quartets and establish whether [5] had provided a theoretical attack that worked in practice. However, upon starting the testing, we were unable to achieve any right quartets, nor were we able to successfully get plaintexts that could meet any of the quartet conditions. Thinking that this would be a similar fix to that of the sandwich attack, we experimented, trying to implement similar debugs, but without success. After a long period spent trying to fix the code, so that it would produce right quartets, without being able to find any issue with what we had written, according to [5], we decided to assume a 'worst-case scenario' and so then started to test the code under the assumption that the paper was wrong. We were able to take this assumption because of the vigorous testing we performed on the implementation of GOST via KATs, which meant we knew there was no issue with that part of the implementation. We did this by performing the attack half in reverse. What this means is that we still used plaintexts P_a and P_b as before and encrypted them to obtain ciphertexts C_a and C_b respectively however, rather than then getting two new ciphertexts by differing the originals with a known value δ we instead started with two new plaintexts P_c and P_d. What this meant was that we could then encrypt all four plaintexts using their respective keys to get four ciphertexts and from this we could then see the

most common differences between the pairs of ciphertexts, (C_a, C_c) and (C_b, C_d), and whether this lined up with the value for δ provided by [5]. What we found was that the most probable differences for this stage of the attack were not the same as the provided value for δ from [5]. Figure 5 shows the results from the testing described above, notably it records the number of times certain differentials occurred in $10,000$ tests. What we found using this data was that not only was the proposed differential from [5] not present, but the most probable differential was occurring with probability $\sim 2^{-2}$. This is a significant increase compared to the original differential, which was supposedly occurring with probability 2^{-6}. Hence we decided to change our value for the δ differential to $0x0000040000000000$ (This value appears different to the one in Figure 5 but that is because the values in Figure 5 are not zero padded), based on the experimental data we had just collected, from the original value of $0x0000008000000000$ as presented in [5]. This should mean that we would now be able to have ciphertexts that, when differed by this new δ value, would produce plaintexts that adhere to the related-key boomerang quartet conditions.

Differential	0x18000000000	0x68000000000	0x70000000000	0x40000000000	0x10000000000	0x60000000000
Number of occurrences in 10,000 tests	1265	1244	2467	2553	1204	1267

Fig. 5. A table showing the number of occurrences in $10,000$ tests for δ differentials within the related-key boomerang attack on GOST

After changing the value of δ, we were then able to achieve right quartets and so had managed to implement a working related-key boomerang attack. However, we were unsure as to why the differentials provided in [5] did not work, as it appears as though they are a somewhat common choice of differential among related-key boomerang attacks, notably [36] and [7] both use the same differentials provided in [5]. Why these particular differentials did not work for this project, we are unsure, but given our scepticism surrounding the validity of [5], it would not be unreasonable to suggest that there are issues present within that paper that we were previously unaware of. Due to the issues we faced surrounding the incorrect/non-working differentials, the project was put further behind schedule as it took far longer to implement the related-key boomerang attack on GOST than initially predicted.

3.3 A Sandwich Attack On GOST

With both of the building blocks complete, we were next tasked with combining them to form a sandwich attack on GOST. However, due to the expected complexities that creating a sandwich attack on GOST would entail, it was planned that we would first try to prove theoretically if it was possible and efficient before attempting an implementation.

The first problem we faced was where we were going to put the layer that incorporated the sandwich distinguisher. Due to the nature of GOST being split into two unequal halves (the first 24 rounds and the last 8) it seemed logical that the layer could be added in one of two places: at round 24 or round 25 (the start of the last 8 rounds). By analysing each round, we were able to establish that the key schedule of GOST for round 25 is the inverse of the key schedule for round 24. What this means is that the key applied at round 24 is cancelled out by the key applied in round 25. Because the keys for these rounds cancel out, it means that we are able to add the filling layer to round 25 as with no addition of keys, it makes this round a simple Feistel switch, the same as in the KASUMI sandwich distinguisher. What this then means is that we are able to achieve an output differential for round 25 of: $\beta = (e_{31}, 0)$, with probability 1. By working through a full diagram of the GOST block cipher we were able to prove theoretically that by adding the filling at layer 25 we did in fact receive the expected output differential with probability 1 and hence establishing that a sandwich attack on GOST would be feasible and that it would be at least as efficient as the related-key boomerang attack on GOST.

4 Results

In order to establish whether the methods implemented in this project are fully functional and are able to achieve similar results to the papers they originated from, we conducted a number of tests. Specifically, we show that the implementations in the project can obtain a reasonably similar or greater number of expected quartets for a given number of trials. We also show that the theory we created surrounding the sandwich attack on GOST should result in a more efficient attack compared to the related-key boomerang attack on GOST from [5].

4.1 KASUMI and The Sandwich Attack

Within [1], Figure 6 is presented showcasing the number of right quartets obtained for each of the $100,000$ experiments that were run. These results are then compared to the expected number of right quartets, which is predicted to follow a Poisson distribution with a mean value of eight.

In order to obtain these results, the following method is presented in [1]: for each experiment, generate a random quartet of keys that satisfy the relevant key differences. Then generate 2^{16} plaintext quartets following the sandwich procedure. Test every plaintext quartet against the sandwich quartet conditions and note the number of quartets that meet all conditions, i.e. a right quartet. This process is then to be repeated $100,000$ times. Alongside the obtained results [1] also generates the number of expected right quartets using the Poisson distribution in order to compare with their results and verify the implementation's correctness. From the probability analysis undertaken in [1], it was established that the probability of the related-key sandwich distinguisher (the probability of a random quartet being a right quartet) is 2^{-14}. So, given that each experiment contained 2^{16} randomly generated quartets the expected number of right quartets (for one test), and hence the mean value used for the Poisson distribution, is calculated as $2^{16} \cdot 2^{-14} = 4$. However, [1] utilises a slight improvement to the first differential suggested in [42]. This improvement to the first differential fixes the value of two plaintext bits and so increases the probability of it occurring by a factor of 2 in the encryption direction. Hence, the expected number of right quartets is then calculated as $2^{16} \cdot 2^{-14} \cdot 2 = 8$ and this

value is used as the mean in the Poisson distribution. As can be seen from Figure 6, the experimental results achieved in [1] closely follow the expected results. So, in order to verify that the implementation in this project was successful, we would need to perform a similar experiment and confirm that it provided values that were close to the expected.

Right quartets	0	1	2	3	4	5	6	7	8
Theory (*Poi*(8))	34	268	1,073	2,863	5,725	9,160	12,214	13,959	13,959
Experiment	32	259	1,094	2,861	5,773	9,166	12,407	13,960	13,956

Right quartets	9	10	11	12	13	14	15	16	17
Theory (*Poi*(8))	12,408	9,926	7,219	4,813	2,962	1,692	903	451	212
Experiment	12,230	9,839	7,218	4,804	3,023	1,672	859	472	219

Right quartets	18	19	20	21	22	23	24	25
Theory (*Poi*(8))	94	40	16	6	2	0.8	0.26	0.082
Experiment	89	39	13	12	2	0	0	1

Fig. 6. A table showing the number of achieved right quartets against the number of expected right quartets for a sandwich attack implemented on KASUMI with $100,000$ experiments from [1]

The implementation of this test into the project was relatively simple to do, as it mostly revolved around allowing the sandwich attack to run for more than just one iteration of plaintext generation. As mentioned above, [1] implemented an improvement to the differentials which resulted in the probability of a right quartet occurring increasing by a factor of two however, we elected not to incorporate this improvement as we felt that it was an unnecessary addition to the project that would add too much complexity. As such, in order to calculate the expected number of right quartets, we would need to use the original value for the related-key sandwich distinguisher of four as the mean for the Poisson distribution. Also, due to time restraints surrounding this project, we have only tested the implementation in this project for $10,000$ experiments rather than the $100,000$ experiments as used in [1]. The results for this test are summarised in Figure 7, and it can be seen that the experimental results achieved for the implementation in this project closely follow the expected results to a reasonable degree of random error. Despite the significant reduction in the number of experiments performed, we are still able to get results that suggest the implementation of the sandwich attack on KASUMI is correct and performs as expected. It would also be reasonable to suggest that if we were to have run our test for the full $100,000$ experiments, then the results achieved would have been similarly close or closer to the expected values. We also believe that although the differential improvement was not incorporated into the project, it would not have affected the spread of results, and we still would be able to obtain results that closely follow the expected. Despite this, by not incorporating the differential improvement, it does mean that we do not have a direct comparison to the results from [1], potentially suggesting that there may be some slight differences that have gone unnoticed.

4.2 GOST and The Related-Key Boomerang Attack

Despite containing a theoretical attack, originally presented in [40], [5] claims that the theory behind this attack is wrong and yet does not go on to experimentally verify the original

Right Quartets	0	1	2	3	4	5	6	7
Theory (Poi{4})	183	733	1465	1954	1954	1563	1042	595
Experiment	185	873	1446	1964	1919	1623	1007	550

Right Quartets	8	9	10	11	12	13	14	15
Theory (Poi{4})	298	132	53	19	6	2	1	0
Experiment	304	129	45	18	13	3	2	0

Fig. 7. The table showing the number of achieved right quartets against the number of expected right quartets for the implemented sandwich attack on KASUMI with $10,000$ experiments

attack. As such, we provide the first experimental verification of the attack originally presented in [40]. However, what this means is that there is no evidence to compare the results from this project to, and therefore, we have no way of validating that we have successfully implemented the related-key boomerang attack on GOST. However, as has already been established, in this implementation we have had to use different differentials due to not being able to get the ones presented in [5] to work. So, it is likely that the results we achieved would be different to those that could have been obtained from an implementation using the original differentials. Despite this, we decided that we should still perform a test similar to the one used for the sandwich attack against KASUMI so as to try and see if the probability of the related-key boomerang distinguisher (the probability of a quartet being a right quartet) is correct.

It is suggested in [5] that the probability of finding a related-key boomerang quartet is 2^{-6}. So, to try draw some similarities to the sandwich attack on KASUMI we decided to perform $10,000$ experiments each utilising 2^8 randomly generated plaintext quartets as this meant that we could utilise a Poisson distribution with a mean value of 4 like the sandwich attack on KASUMI. This would at least allow for some kind of verification to take place, as we would be able to compare these new results to the ones obtained in the sandwich attack on KASUMI, hopefully providing a sense of correctness to the results we achieved. It was at this point that we neglected to think about the fact that we had changed the differentials for the related-key boomerang attack, and so using the probability of 2^{-6} was not correct. We instead should have utilised the value we calculated for the probability of the new differentials, which was 2^{-2}, and hence we then should have used 2^6 plaintexts in order to achieve a mean value of 4. Despite this crucial point being missed, the rest of the testing then followed the same procedure as the one used for the sandwich attack, with the only difference being that we utilised the related-key boomerang quartet conditions instead of the sandwich ones. The results table from this test is summarised in Figure 8.

As can be seen from Figure 8, the number of right quartets achieved in the testing was significantly higher than expected. This has meant that the expected number of right quartets from the Poisson distribution with a mean of 4 does not line up with the experimental values. Although we speculated that we may achieve slightly different values from what was expected due to using a different differential from the one used in [5], we were not expecting this drastic of a difference in the values. Also included in Figure 8 is the expected number of right quartets for $10,000$ experiments using a Poisson distribution with a mean value of 15. Having looked at the results we achieved, this new

Poisson distribution is what we thought to be the best fit for the experimental data. Although reasonably close to the expected values, the obtained results do differ more than we would like for it to be considered a good match. However, we thought that by running more than $10,000$ tests then the results would converge to the expected values more closely. So, we reran the experiment this time using $100,000$ test cases. The results table for this experiment is summarised in Figure 9. As can be seen from the data in Figure 9, the results actually appear to be more diverged from the expected values compared to the results from the experiment with $10,000$ tests. This goes against what we were expecting and may suggest that we are using an incorrect value for the mean used in the Poisson distribution.

Right Quartets	2	3	4	5	6	7	8	9
Theory (Poi{4})	1465	1954	1954	1563	1042	595	298	132
Experiment	0	1	1	8	21	42	79	157
Theory (Poi{15})	0	2	7	19	48	104	194	324

Right Quartets	10	11	12	13	14	15	16	17
Theory (Poi{4})	53	19	6	2	1	0	0	0
Experiment	303	436	618	776	887	1039	1024	993
Theory (Poi{15})	486	663	829	956	1024	1024	960	847

Right Quartets	18	19	20	21	22	23	24	25
Theory (Poi{4})	0	0	0	0	0	0	0	0
Experiment	878	742	594	438	364	235	126	109
Theory (Poi{15})	706	558	418	299	204	133	83	50

Right Quartets	26	27	28	29	30	31	32	33
Theory (Poi{4})	0	0	0	0	0	0	0	0
Experiment	66	28	15	9	7	2	2	0
Theory (Poi{15})	29	16	9	4	2	1	1	0

Fig. 8. The table showing the number of achieved right quartets, the number of expected right quartets and the new number of expected quartets for the implemented related-key boomerang attack on GOST with $10,000$ experiments

Right Quartets	2	3	4	5	6	7	8	9
Experiment	2	3	21	57	185	418	929	1595
Theory (Poi{15})	3	17	65	194	484	1037	1944	3241

Right Quartets	10	11	12	13	14	15	16	17
Experiment	2901	4404	5949	7792	9202	10025	10344	9747
Theory (Poi{15})	4861	6629	8286	9561	10244	10244	9603	8474

Right Quartets	18	19	20	21	22	23	24	25
Experiment	8933	7427	6040	4628	3361	2260	1545	950
Theory (Poi{15})	7061	5575	4181	2987	2036	1328	830	498

Right Quartets	26	27	28	29	30	31	32	33
Experiment	543	348	193	93	55	29	12	4
Theory (Poi{15})	287	160	86	44	22	11	5	2

Right Quartets	34	35	36	37	38	39	40	41
Experiment	3	1	1	0	0	0	0	0
Theory (Poi{15})	1	0	0	0	0	0	0	0

Fig. 9. The table showing the number of achieved right quartets and the number of expected right quartets for the implemented related-key boomerang attack on GOST with $100,000$ experiments

To try and establish why the results we were getting were so diverged from the expected values we generated the graphs in Figure 10 and Figure 11 showing the results from the experiment with $10,000$ and $100,000$ tests respectively. Comparing the two figures, we can see that the proposal we made, that running more tests should allow the experimental results to converge more to the expected results, was not strictly wrong as the spread of results in Figure 11 do more closely follow the bell curve shape of a Poisson distribution compared to Figure 10. However, we can also now see more clearly that both sets of data appear right-skewed in comparison to a Poisson distribution with a mean of 15. Although in Figure 10 the greatest number of right quartet occurrences is for a value of 15, in Figure 11 the greatest number of occurrences is for a value of 16. This suggests that using a Poisson distribution with a slightly higher mean value might fit the data better and present a less skewed graph. As such, we have included a Poisson distribution with a mean value of 16 onto both graphs, and this fits the data much better than the original value of 15.

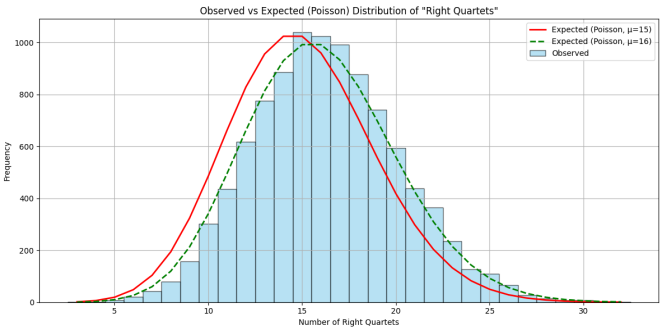

Fig. 10. A graph showing the number of achieved right quartets against a Poisson distribution with mean 15 and mean 16 for the implemented related-key boomerang attack on GOST with $10,000$ experiments

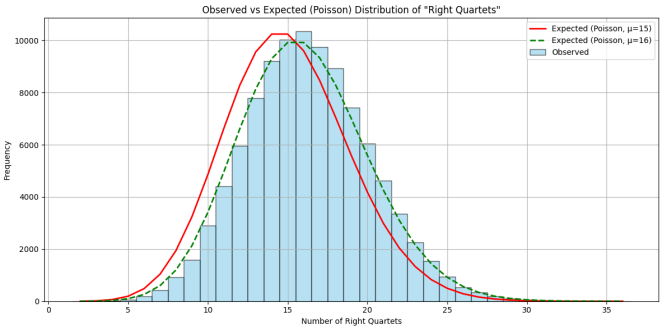

Fig. 11. A graph showing the number of achieved right quartets against a Poisson distribution with mean 15 and mean 16 for the implemented related-key boomerang attack on GOST with $100,000$ experiments

Using this newly calculated mean value of 16, we next needed to check the true probability of a right quartet occurring within our experiment. Taking the equation from before, rearranging it and substituting in the experimental values, we get:

$$16 \div 2^8 = P$$

Where 16 is the new mean value established for the Poisson distribution, 2^8 is the number of randomly chosen quartets per experiment and P is the probability of the related-key boomerang distinguisher. This gives a value for P of:

$$P = \frac{16}{256} = \frac{1}{16} = 0.0625 = 2^{-4} \neq 2^{-2}$$

This value of P suggests that the probability we calculated for our new differentials occurring is, in fact, wrong, and we are off by a factor of two. We are unsure why this has happened and would need to perform more tests to confirm whether the values we have are correct. However, by using the new differential, we have been able to increase the probability of a right quartet occurring by a factor of

4 compared to the probability from [5]. Although we are sceptical about this value we have been unable to find evidence to suggest that within the experiment we achieve a different probability to 2^{-4} and we have been unable to find a reason to believe that the expected value of 2^{-2} is incorrect. However, we believe that it is probably worth applying some more rigorous tests outside of this project to establish why we were unable to replicate the results from [5] and subsequently why the new differential we implemented appears not to provide the probability of a right quartet occurring as expected.

4.3 GOST and The Sandwich Attack

With the creation of our new attack, we needed to be able to show that in theory it would work as well as the related-key boomerang attack on GOST from [5]. We did this by analysing the three cascading sub-ciphers that we created as part of the sandwich distinguisher. The first sub-cipher is comprised of the first 24 rounds of GOST and remains unchanged. This means that any input differentials passed through these rounds will result in specific output differentials with the same probability as with the related-key boomerang attack on GOST. The next sub-cipher is the filling layer on round 25, which we were able to establish occurs with probability 1. Due to the nature of the related-key boomerang attack on GOST, we cannot make a direct comparison with this new filling layer; however, because it occurs with probability 1, we can say that this is not worse than the related-key boomerang attack. The final sub-cipher we created is the final 7 rounds of GOST. This differs from the related-key boomerang distinguisher, which uses the last 8 rounds. This difference occurs because of the addition of the filling layer at round 25, essentially taking a round away from the last sub-cipher. Although we did not find the exact probability of this new distinguisher, we were able to decipher that it would have a greater probability than the previous 8 round distinguisher. This is because the lower number of rounds results in specific output differentials appearing with higher probability. So, from our work, we are able to deduce that the new sandwich distinguisher for GOST should have a probability at least as good as (but likely higher than) the related-key boomerang distinguisher for GOST. This deduction answers the research question, as we have been able to display that the sandwich attack is a feasible method for breaking the GOST block cipher.

5 Evaluation

We now look back on the research question: 'Is the sandwich attack a feasible method of breaking the GOST block cipher?' We answer this by examining the preliminary work and how its strengths and weaknesses helped to construct an idea of how to perform a sandwich attack on GOST.

5.1 Attacking KASUMI With The Sandwich Distinguisher

The first basic deliverable called for an implementation of a sandwich attack on the KASUMI block cipher, attempting to replicate the work and results from [1]. This was accomplished relatively early on in the project lifecycle, although it was delayed due to previously mentioned issues. We have been able to show the accomplishment of this deliverable by presenting the data in Figure 7, which closely matches the expected results determined in [1]. Although the algorithms used within this project have not been optimised for efficiency, we are still able to achieve right quartets for the sandwich distinguisher quickly (on average, a single right quartet can be found in under thirty seconds). Despite the setbacks experienced with this section of development, the implementation of the sandwich attack on KASUMI performs well and was able to achieve the predicted values when tested. However, it is worth re-mentioning that although the implementation in this project works as expected and is reasonably efficient, we did not replicate exactly the version that was implemented in [1]. This was due to not implementing a change to the differentials in this project, and has subsequently meant that the expected value of the sandwich distinguisher in this project is a factor of two lower than in [1]. This, accompanied by the fact that we did not utilise as many test cases as [1], leads to some cause for doubt as to the validity of the implementation in this project, as we have had no exact data to compare to and to verify what we have done. Nevertheless, we believe that the data that has been produced from the implementation in this project is sufficient to say that we have successfully created a simplified version of the algorithm produced in [1]. Also, we can then say that, based on the statistical analysis presented in [1] from which we have generated a set of expected values, the implementation in this project is valid as it produces results sufficiently close to these expected values.

5.2 Attacking GOST With The Related-Key Boomerang Distinguisher

The second basic deliverable called for an implementation of a related-key boomerang attack on the GOST block cipher, attempting to implement the algorithm presented in [5]. This was accomplished later in the project lifecycle than was planned due to a number of issues occurring, which stalled progress. Because we decided to run several KATs on the implementation of GOST in the project, we were sure that the issues were not arising from the implementation of the cipher. However, the issues we encountered resulted in the differentials needing to be changed from those proposed in [5] because we were unable to get those specific differentials to work within our project. Although [5] did not implement the algorithm presented and, as such, had no experimental data which we could use to verify the implementation from this project, we were able to generate a similar test to the one performed for the sandwich attack on GOST. The results from this data, however, did not line up with the values we were expecting based on the theory presented in [5] nor the expected values based on the theory we presented for the new differential. In fact, the solution presented in this project appeared to show an increase by a factor of four for the probability of the related-key boomerang distinguisher, whilst also being a factor of two lower than the value we predicted based on the new differential. Despite this, the methods implemented in this project for the related-key boomerang distinguisher on GOST are reasonably efficient,

with a single right quartet being found consistently in under a second. Given that we were unable to replicate the expected values from [5] within this part of the project, it is reasonable to assume that there exists some inconsistency. Whether that be within the implementation itself or with the original design of the algorithm, we are unsure. However, it is worth noting that despite not being able to replicate [5], we were still able to develop a working implementation, albeit if we are unable to prove its validity.

5.3 Attacking GOST With The Sandwich Distinguisher

With both of the basic deliverables complete, we could then utilise the information we had gained from them to begin the intermediate deliverable, which asked for some form of proof that the sandwich attack would be a feasible method for breaking the GOST block cipher. This section was key to answering the research question. The analysis of how the sandwich distinguisher could be applied to GOST was accomplished very late into the project due to the issues with the basic deliverables. However, we were able to establish that the sandwich distinguisher could be applied to GOST. As such, we were also able to work out that by applying the 'filling' layer to the 25th round of GOST, the probability of the sandwich distinguisher on GOST would be at least as good as the related-key boomerang distinguisher. What this means is that the probability of a sandwich quartet occurring is at least the same as that of a related-key boomerang quartet occurring within its respective distinguisher. This suggests that, in theory, the sandwich attack is a feasible method of breaking the GOST block cipher as it requires no more time, memory or plaintext quartets compared to a related-key boomerang attack on GOST and so answers the research question.

5.4 The Approach

The approach we took towards the project centred heavily around developing existing techniques to use as stepping stones towards the larger goal of trying to answer the research question: 'Is the sandwich attack a feasible method of breaking the GOST block cipher?' For the most part, this worked quite well, developing the sandwich attack on KASUMI from [1] and the related-key boomerang attack on GOST from [5] allowed us to gain a deeper understanding of the attacks. This understanding then helped when deciphering whether the sandwich attack would be applicable to GOST because we were able to utilise techniques and knowledge that had been gained from implementing the attacks previously. As part of this, we were also able to validate previous theoretical attacks experimentally. Specifically, we were able to show that the theoretical attack presented in [5] was a form of correct related-key boomerang attack on GOST, albeit if we needed to adapt the proposal slightly in order for it to work properly. However, we did experience a number of setbacks throughout this project, which hindered the structured development of the solution. Notably, we encountered the majority of the issues within this project when developing the related-key boomerang attack on GOST. These hindrances have meant that some of the more advanced deliverables that we had initially planned to undertake have not been completed. As such, if we were given the opportunity to repeat this work, then we would not change the structure of the approach in terms of replicating attacks to use as a baseline, but would change the focus of it to be more centred around developing new techniques rather than replicating old ones. We would also perform a more in-depth analysis of the papers we wished to replicate to ensure that the number of issues encountered could be lessened. Following on from this, it may also be a good idea to allow more time to complete sections of the project, as this project has run quite a bit over schedule due to unforeseen issues and subsequently has meant that we have been unable to complete everything we set out to do at the start.

6 Conclusion

As the final section of this project, we summarise the methods of the solution that have been presented, discuss the importance of the work undertaken, and also go on to make some suggestions towards future work that could be presented as a follow-up from this project.

6.1 Project Conclusions

To summarise what we have presented in this paper, we first started by replicating one of the few instances of a sandwich attack onto the KASUMI block cipher as presented in [1]. Despite some setbacks we were able to achieve results that were sufficiently similar to those presented in [1] to be able to say that: the sandwich distinguisher is a viable method for providing right quartets for use in an attack and that the results presented in [1] were repeatable.

We then went on to create the first experimental implementation of the related-key boomerang attack presented in [5]. However, we soon came to realise that at least one of the implementations from this project or [5] contained some issues, as we were unable to obtain results that fit the expected theory of either paper. Although after we implemented a change to the differentials used in the related-key boomerang distinguisher, we were able to obtain results that suggest we had improved the probability of this distinguisher by a factor of four, but we were unable to verify these claims as we had no reference data to compare to. What this meant was that we were able to create a related-key boomerang distinguisher that could efficiently identify right quartets, but we were unable to verify this implementation's validity, nor were we able to show that the theory presented in [5] was correct in practice.

Finally, we analysed the possibility of applying the sandwich distinguisher to the GOST block cipher. During this section, we were able to notice that, through the specific placement of the 'filling' layer within GOST, we could apply the sandwich distinguisher to GOST with no extra cost when compared to the related-key boomerang distinguisher. Subsequently this allowed us to answer the proposed research question, 'Is the sandwich attack a feasible method for breaking the GOST block cipher?', by saying that, yes the sandwich attack is a feasible method for breaking the GOST block cipher as it should in theory have no extra requirements compared to a related-key boomerang attack which is an already known and well studied attack. Although

we were unable to progress to creating an implementation of the sandwich attack on GOST, due to the many issues experienced throughout the project, we were still able to successfully answer the research question despite not being able to experimentally verify the proposal.

Despite the many setbacks experienced within this project, we have still been able to contribute a significant amount of work to the cryptographic community. More specifically, we have:

- Verified the theoretical and experimental results of the sandwich distinguisher on KASUMI as presented in [1].
- Created the first experimental verification of the theoretical related-key boomerang attack presented in [5].
- Established inconsistencies within [5] that suggest the proposed attack is not possible, but then go on to modify this attack to ensure it works.
- Provide the first attempt at generalising the sandwich distinguisher for block ciphers other than KASUMI.
- Propose that the sandwich distinguisher is applicable to GOST and should be at least as effective as the related-key boomerang distinguisher.

All of these points could prove to be valuable information in the consideration of the sandwich attack as an established attack method. This is furthering on from the earlier point that the sandwich attack could be used in the same places as a related-key boomerang attack (possibly even replacing it), and hence potentially evolving into a new best-known attack for block ciphers in general. These possibilities enforce the importance of the work undertaken in this project, as we may have developed the beginnings of the next big attack in cryptography.

6.2 Further Work

Based on the findings presented in this project, there are a number of ways in which you could extend the work we have produced, which are summarised below.

The first point that we suggest for further work is to undertake the deliverables for this project that were not completed, namely the implementation of the sandwich distinguisher on GOST. With this implementation, the theory presented in this project could be verified and thus cementing the sandwich distinguisher as a viable method for use against GOST. Also, it would help to validate the claim that the sandwich attack is a feasible method for breaking the GOST block cipher and further answering the research question presented in this project.

As mentioned in the Results section, when implementing the related-key boomerang attack presented in [5], we were unable to replicate the theoretical findings and instead produced an implementation that appears to work with an increased probability that does not match the value we expected based on our theory. As such we think that it would be worth checking the findings we presented to see if they are valid and on top of this verifying whether the theory in [5] is incorrect, as we suspect, or if it is the case that the implementation of the theory from [5] in this project contains some errors.

Another way in which you could build directly on this project would be to implement the full key recovery attacks for both the sandwich and related-key boomerang distinguishers presented in this project. As has already been mentioned, within this project, we have only implemented the distinguishers for each of the proposed attacks rather than the full key recovery variants, as it was thought that the key recovery variants would be too complex and add too much time to the already busy project schedule. As such, an extension of the work in this project would be to implement the key recovery attacks, showing that the distinguishers would work in practice as part of a full attack on the respective block ciphers. Although the full key recovery attack for the sandwich attack on KASUMI was implemented in [1], we believe it would still be worthwhile creating a new implementation of that attack to verify the claims made in [1].

The final way in which we think this work could be extended relates to the process of generalising the sandwich attack. When the sandwich attack was first presented in [1], it was suggested in the 'Future Work' section that finding generic structures in which the sandwich attack is applicable was a key point of interest. In this project, we have taken a step towards this generalisation by showing that the sandwich attack can be applied to GOST in theory, but we did not go on to provide any further generalisation. As such, if the sandwich attack could be fully generalised and then subsequently applied to ciphers such as AES, we could see an improvement in the best-known attacks for some of the strongest ciphers that currently exist.

We conclude this paper with a formal statement of the directions for further research raised above.

- Problem 1: Develop an implementation for the sandwich distinguisher on GOST.
- Problem 2: Attempt to verify the claims made in this project surrounding the related-key boomerang distinguisher on GOST and then go on to agree or disagree with the findings presented about the theory from [5].
- Problem 3: Develop implementations for the full key recovery attacks of the distinguishers presented in this project.
- Problem 4: Attempt to further generalise the sandwich attack for use on generic block ciphers.

REFERENCES

[1] O. Dunkelman, N. Keller, and A. Shamir, "A practical-time related-key attack on the KASUMI cryptosystem used in GSM and 3G telephony," *Journal of Cryptology*, vol. 27, pp. 824–849, Oct. 2014.

[2] M. Gorski and S. Lucks, "New related-key boomerang attacks on AES." Cryptology ePrint Archive, Report 2008/438, 2008.

[3] A. Biryukov and D. Khovratovich, "Related-key cryptanalysis of the full AES-192 and AES-256." Cryptology ePrint Archive, Report 2009/317, 2009.

[4] P. Derbez, M. Euler, P.-A. Fouque, and P. H. Nguyen, "Revisiting related-key boomerang attacks on AES using computer-aided tool," in *Advances in Cryptology – ASIACRYPT 2022, Part III* (S. Agrawal and D. Lin, eds.), vol. 13793 of *Lecture Notes in Computer Science*, pp. 68–88, Springer, Cham, Dec. 2022.

[5] V. Rudskoy, "On zero practical significance of "key recovery attack on full GOST block cipher with zero time and memory"." Cryptology ePrint Archive, Report 2010/111, 2010.

[6] W. J. Buchanan, "Kasumi - a5/3 - cipher." https://asecuritysite.com/symmetric/kasumi, 2025. Accessed: April 13, 2025.

[7] M. Pudovkina and G. Khoruzhenko, "Related-key attacks on the full gost block cipher with 2 or 4 related keys," Apr 2019.

[8] "GitHub - bozhu/GOST-Python: A Python implementation for the block cipher GOST — github.com." https://github.com/bozhu/GOST-Python?tab=readme-ov-file#, 2014. [Accessed 13-04-2025].

[9] V. Nachef, J. Patarin, and E. Volte, *Feistel ciphers*. Springer, 2017.

[10] J. Jean, "TikZ for Cryptographers." https://www.iacr.org/authors/tikz/, Feb 2015.

[11] D. R. Anderson, *Information Theory and Entropy*. New York, NY: Springer New York, 2008.

[12] C. Carlet, *Vectorial Boolean Functions for Cryptography*. Encyclopedia of Mathematics and its Applications, Cambridge University Press, 2010.

[13] D. Coppersmith, "The data encryption standard (des) and its strength against attacks," *IBM Journal of Research and Development*, vol. 38, no. 3, 1994.

[14] R. Avanzi, "A salad of block ciphers." Cryptology ePrint Archive, Report 2016/1171, 2016.

[15] F. Pub, "Data encryption standard (des)," *FIPS PUB*, vol. 463, 1999.

[16] E. F. Foundation, M. Loukides, and J. Gilmore, *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design*. USA: O'Reilly & Associates, Inc., 1998.

[17] J. Daemen and V. Rijmen, "Aes proposal: Rijndael," 1999.

[18] E. Barker, L. Feldman, and G. Witte, "Guidance on the tdea block ciphers," 2017-11-27 2017.

[19] V. Dolmatov, "GOST 28147-89: Encryption, Decryption, and Message Authentication Code (MAC) Algorithms." RFC 5830, Mar. 2010.

[20] V. Dolmatov, "GOST R 34.12-2015: Block Cipher "Kuznyechik"." RFC 7801, Mar. 2016.

[21] N. T. Courtois, "Security evaluation of GOST 28147-89 in view of international standardisation." Cryptology ePrint Archive, Report 2011/211, 2011.

[22] V. Dolmatov and D. Baryshkov, "GOST R 34.12-2015: Block Cipher "Magma"." RFC 8891, Sept. 2020.

[23] 3GPP, "3gpp ts 35.202 v18.0.0 (2024-03) technical specification 3rd generation partnership project; technical specification group services and system aspects; 3g security; specification of the 3gpp confidentiality and integrity algorithms; document 2: Kasumi specification (release 18)."

[24] M. Matsui, "New block encryption algorithm MISTY," in *Fast Software Encryption – FSE'97* (E. Biham, ed.), vol. 1267 of *Lecture Notes in Computer Science*, pp. 54–68, Springer, Berlin, Heidelberg, Jan. 1997.

[25] A. Biryukov, "Block ciphers and stream ciphers: The state of the art." Cryptology ePrint Archive, Report 2004/094, 2004.

[26] J. Katz and Y. Lindell, *Introduction to modern cryptography*. Chapman & Hall/CRC Cryptography and Network Security Series, Philadelphia, PA: Chapman & Hall/CRC, Aug. 2007.

[27] T. Sale, "The bletchley park 1944 cryptographic dictionary formatted by tony sale (c) 2001," 2001.

[28] O. Singh, "Known Plaintext-Attacks-Explained — cointelegraph.com." https://cointelegraph.com/explained/known-plaintext-attacks-explained, 2024. [Accessed 15-04-2025].

[29] E. Biham, "New types of cryptanalytic attacks using related keys," *Journal of Cryptology*, vol. 7, pp. 229–246, Dec. 1994.

[30] H. van Tilborg and S. Jajodia, *Encyclopedia of Cryptography and Security*. No. v. 1 in Encyclopedia of Cryptography and Security, Springer, 2011.

[31] E. Biham and A. Shamir, "Differential cryptanalysis of DES-like cryptosystems," *Journal of Cryptology*, vol. 4, pp. 3–72, Jan. 1991.

[32] D. Wagner, "The boomerang attack," in *Fast Software Encryption – FSE'99* (L. R. Knudsen, ed.), vol. 1636 of *Lecture Notes in Computer Science*, pp. 156–170, Springer, Berlin, Heidelberg, Mar. 1999.

[33] E. Biham, O. Dunkelman, and N. Keller, "Related-key boomerang and rectangle attacks," in *Advances in Cryptology – EUROCRYPT 2005* (R. Cramer, ed.), vol. 3494 of *Lecture Notes in Computer Science*, pp. 507–525, Springer, Berlin, Heidelberg, May 2005.

[34] A. Biryukov and D. Khovratovich, "Related-key cryptanalysis of the full AES-192 and AES-256," in *Advances in Cryptology – ASIACRYPT 2009* (M. Matsui, ed.), vol. 5912 of *Lecture Notes in Computer Science*, pp. 1–18, Springer, Berlin, Heidelberg, Dec. 2009.

[35] O. Dunkelman, N. Keller, and A. Weizman, "Practical-time related-key attack on GOST with secret S-boxes," in *Advances in Cryptology – CRYPTO 2023, Part III* (H. Handschuh and A. Lysyanskaya, eds.), vol. 14083 of *Lecture Notes in Computer Science*, pp. 177–208, Springer, Cham, Aug. 2023.

[36] Y. Ko, S. Hong, W. Lee, S. Lee, and J.-S. Kang, "Related key differential attacks on 27 rounds of XTEA and full-round GOST," in *Fast Software Encryption – FSE 2004* (B. K. Roy and W. Meier, eds.), vol. 3017 of *Lecture Notes in Computer Science*, pp. 299–316, Springer, Berlin, Heidelberg, Feb. 2004.

[37] M.A.Pudovkina and G.I.Khoruzenko, "An attack on the GOST 28147-89 block cipher with 12 related keys (Russian)," *Mathematical Aspects of Cryptography*, vol. 4, pp. 127–152, Jan 2013.

[38] O. Dunkelman, N. Keller, and A. Shamir, "A practical-time related-key attack on the KASUMI cryptosystem used in GSM and 3G telephony," in *Advances in Cryptology – CRYPTO 2010* (T. Rabin, ed.), vol. 6223 of *Lecture Notes in Computer Science*, pp. 393–410, Springer, Berlin, Heidelberg, Aug. 2010.

[39] A. Jana, M. Rahman, D. Saha, and G. Paul, "Switching the top slice of the sandwich with extra filling yields a stronger boomerang for NLFSR-based block ciphers." Cryptology ePrint Archive, Report 2023/1543, 2023.

[40] E. Fleischmann, M. Gorski, J.-H. Huhne, and S. Lucks, "Key recovery attack on full gost block cipher with zero time and memory," 2009.

[41] B. Schneier, *Still Other Block Ciphers*, ch. 14, pp. 331–355. John Wiley & Sons, Ltd, 2015.

[42] M. Blunden and A. Escott, "Related key attacks on reduced round KASUMI," in *Fast Software Encryption – FSE 2001* (M. Matsui, ed.), vol. 2355 of *Lecture Notes in Computer Science*, pp. 277–285, Springer, Berlin, Heidelberg, Apr. 2002.

[43] P. Zajac and R. Čagala, "Local reduction and the algebraic cryptanalysis of the block cipher gost," *Periodica Mathematica Hungarica*, vol. 65, 12 2012.

APPENDIX A
KASUMI

Four figures showing diagrams of how the functions that make up the KASUMI block cipher work from [23].

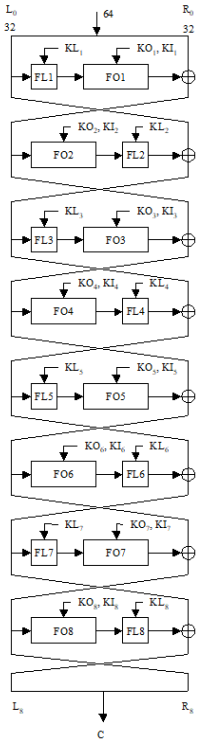

Fig. 12. A figure showing the workings of the main KASUMI function

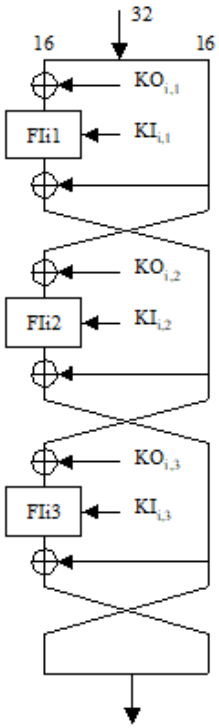

Fig. 13. A figure showing the workings of the FO function from KASUMI

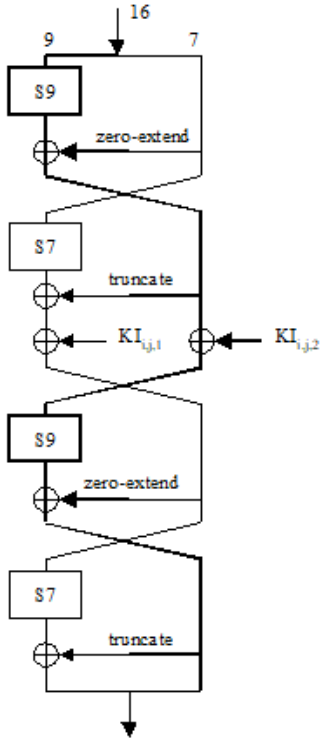

Fig. 14. A figure showing the workings of the FI function from KASUMI

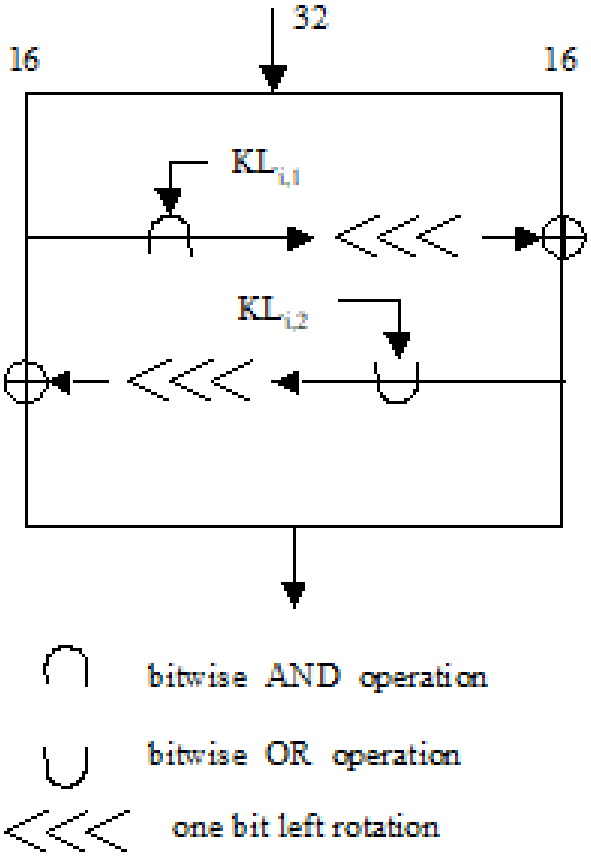

Fig. 15. A figure showing the workings of the FL function from KASUMI

APPENDIX B
GOST

Figures showing a diagram of one round and 32 rounds of the GOST block cipher.

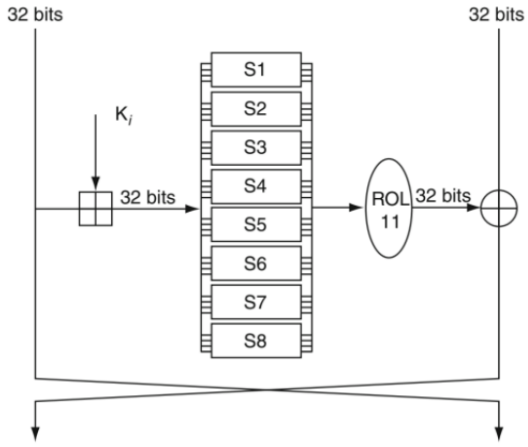

Fig. 16. A figure showing the workings of one round of GOST from [30, pp. 517]

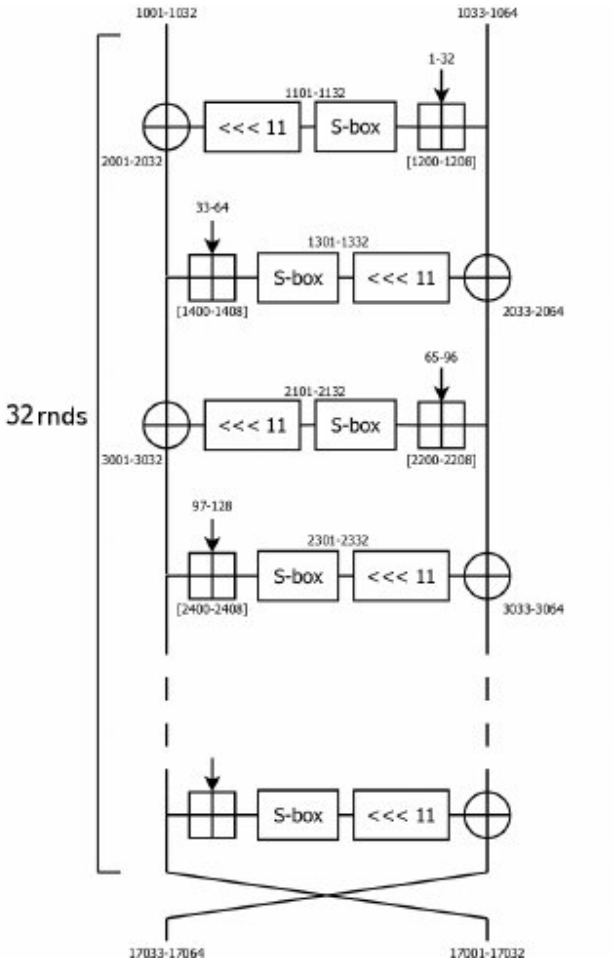

Fig. 17. A figure showing the workings of 32 rounds of GOST from [43]