Big Brother Is Watching You: A Closer Look At Backdoor Construction

Anubhab Baksi[1], Arghya Bhattacharjee[2], Jakub Breier[3], Takanori Isobe[4], and Mridul Nandi[2]

[1] Nanyang Technological University, Singapore

[2] Indian Statistical Institute, Kolkata, India

[3] Silicon Austria Labs, Graz, Austria

[4] University of Hyogo, Kobe, Japan

anubhab001@e.ntu.edu.sg, bhattacharjeearghya29@gmail.com, jbreier@jbreier.com, takanori.isobe@ai.u-hyogo.ac.jp, mridul.nandi@gmail.com

Abstract. With the advent of Malicious (Peyrin and Wang, Crypto'20), the question of a cipher with an intentional weakness which is only known to its designer has gained its momentum. In their work, the authors discuss how an otherwise secure cipher can be broken by its designer with the help of a secret backdoor (which is not known to the user/attacker). The contribution of Malicious is to propose a cipher-level construction with a backdoor, where it is computationally infeasible to retrieve the backdoor entry despite knowing how the mechanism works.
In this work, we revisit the work done by Peyrin and Wang in a greater depth. We discuss the relevant aspects with more clarity, thereby addressing some of the important issues connected to a backdoor construction. The main contribution, however, comes as a new proof-of-concept block cipher with an innate backdoor, named ZUGZWANG. Unlike Malicious, which needs new/experimental concepts like partially non-linear layer; our cipher entirely relies on concepts which are well-established for decades (such as, using a one-way function as a Feistel cipher's state-update), and also offers quite a few advantages over Malicious (easy to visualise, succeeds with probability 1, and so on). Having known the secret backdoor entry, one can recover the secret key with only 1 plaintext query to our cipher; but it is secure otherwise. As the icing on the cake, we show the provable security claims for our cipher.

Keywords: backdoor · hash function · xof · block cipher · feistel · low-mc · malicious · low-mc-m · provable security · sprp · white-box

1 Introduction

One of the problems that comes with designing a cipher is to gain the collective trust of the community. The cipher must satisfy certain security requirement with sufficient margin to prevent a malicious attacker (who has the full knowledge of the cipher specification) from getting information secured by the cipher under a secret key. At the same time, it is also essential that the cipher designer will fail to retrieve the data secured by the cipher under a secret key. Stated in other words, the designers of a cipher have to convince the rest of the community that the cipher does not have a hidden vulnerability that evades known cryptanalytic methods (thus, it is known only to the designers). As we have seen, this is not always the situation. Case in point, it has long been speculated that the SIMON and SPECK [3] family of block ciphers have some form of hidden backdoor (see [19] or Schneier's blog[5]. among other sources[6]), which

[5] https://www.schneier.com/blog/archives/2018/04/two_nsa_algorit.html

[6] It is also worth pointing out that problem is partly exacerbated due to the absence of any cryptanalytic result in the introducing paper [3].

are only known to the designers[7]. Despite years of speculation, the presence of any backdoor is not determined.

Amidst such situation, it is not surprising that the cryptographic community will take interest in the prospect of designing a cipher with an implanted backdoor. We have recently seen this happening in the Crypto'20/Eprint'20 paper [19] where the designers take an otherwise secure cipher family and implant a backdoor in it. They present their contribution in the form of a framework, named, `Malicious`. It works by querying the cipher with a chosen tweak difference on a variant of the `LOWMC` [1] family of ciphers (this tweak difference is secret and known only by the cipher designer). Ultimately, this tweak difference propagates through the cipher in such a way that the resulting ciphertext difference allows the cipher designer to retrieve the secret key (the secret key is chosen by, and only known to the user) with a certain probability. They also describe a `Malicious` based tweakable block cipher, named `LOWMC-M`.

1.1 Contribution

A big part of the inspiration of our work goes to the Crypto'20 paper by Peyrin and Wang [19]. More precisely, we take a deeper look at the `Malicious` framework (and its instance `LOWMC-M`), and improve the state-of-the-art in a number of ways.

To begin with, we show a provably secure construction of backdoor that improves from `LOWMC-M` [19]. Our method of the backdoor construction relies entirely on pre-existing notions of security, which are well-known/well-analysed for decades. This starkly contrasts with the new/experimental construction of `Malicious`, that relies on lesser studied concepts such as partially non-linear layer. Apart from that, our backdoor requires only 1 plaintext query (works with probability 1), unlike the `LOWMC-M` that requires a number of chosen (plaintext, tweak) queries. We do not need any tweak, and the overall idea is generic – it can be implemented atop virtually any encryption and hash algorithm[8]. Thus, making it possible to have a backdoor without any tweak and not tied to `LOWMC` [1].

The coverage/contribution of our paper does not end there. We ask several relevant questions, which have not been answered yet. We argue that no matter how cleverly the backdoor is designed, it is not possible for the designer to access it without the user's cooperation (as the user can always cross-check if some secret information is revealed – and if so – can deny the request); or one backdoor entry cannot be used more than once (as the attacker will get to know as soon as it is used). The elephant in the room, however, lurks in hiding the key which is released as a result of the backdoor access — the key is not encrypted in any way, meaning the attacker gets to know about it no later than the designer does.

1.2 Prerequisite

As discussed in [19, Section 1], the concept of backdoor itself is not new. In our context, we directly follow [19]. For clarity, the terms/ideas used are briefly described here.

The cipher designer, whom we refer to as *Derek* for simplicity, designs a cipher with an intentional backdoor (which is known only by him). The cipher (the public description of the

[7] In this case, the designers are a group of researchers from the American government's National Security Agency (NSA), possibly hinting at a government-level initiative in the background.

[8] Depending on the hash output size and the state size of the encryption algorithm, we may need to pad/truncate.

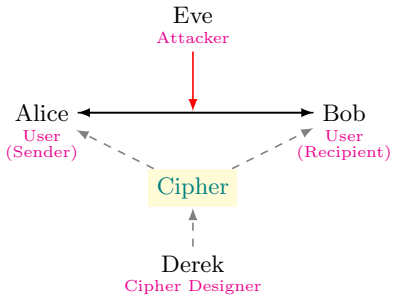

Fig. 1: Schematic of backdoor work-flow

cipher, to be more precise) is then used by the users, Alice and Bob, to communicate sensitive information. The attacker, Eve, watches the channel between Alice and Bob closely and knows all the (publicly available) specification/cryptanalysis regarding the cipher. Figure 1 shows a schematic representation.

Now, at some point during communication, Derek can use the backdoor to retrieve sensitive information, this incidence (if happens) is indicated by backdoor *access*. The backdoor *mechanism* lets Derek to access sensitive information (this works as a weakened version of the cipher). The mechanism is activated with the help of a backdoor *entry* (e.g., a 128-bit string when used as the plaintext to the cipher), which is known only to Derek (it will likely become a public knowledge after it has been used once).

For the interest of brevity, we assume the reader's familiarity with the basic terms/concepts, including; CSPRNG, LFSR, block cipher (along with padding, and mode of operation like `CTR`), stream cipher (along with IV and nonce), hash function, MAC, AE, AEAD; PRP, SPRP; OWF; cipher families (Feistel, SPN and ARX); and ciphers (`DES`, `AES`, `RC4`, `RSA`). We also use XOF[9] (eXtended Output Function).

1.3 Organisation

The background information is covered in Section 2 (particularly Section 2.2 contains some previously unreported observations). Section 3.1 goes through the practical aspects of a backdoor, and Section 3.2 covers two related notions of security.

In Section 4, we present our Feistel network based block cipher named "`ZUGZWANG`"[10] that has a backdoor[11]. After the fundamental idea is stated in Section 4.1, we show a concrete instance by using `AES-128` and `SHAKE-128`[9] in Section 4.2, followed by the provable security claim which can be found in Section B. Apart from that, a comparison with `Malicious` is given in Section 4.3.

The conclusion can be found in Section 5. Some postscript thoughts are given in Section A. Finally in Section C, we present some test cases for the concrete instance of `ZUGZWANG`.

[9] See https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf.

[10] It is a German word (translates to 'a compulsion to move'), used in context of Chess to describe wherein all the available moves for a player make the situation worse.

[11] As it has a backdoor, any practical application of `ZUGZWANG` is not recommended (to be used mostly, if not only, as an interesting proof-of-concept).

2 Background

2.1 Implementation Level and Cipher Level Backdoors

The term, 'backdoor' is generally more common in the cyber-security or hacking communities. Here it typically refers to an intentionally implanted weakness[12]. This process is done at the fabrication/implementation level and transparent at the cipher design level[13]. In other words, it is possible that the cipher being used does not have any potential security flaw, but it the call to the cipher is be bypassed by using a secret mechanism. This mechanism is generally too small compared to the rest of the circuit that it evades during inspection, or is made obscure by some implementation/protocol level technique. Most of the time the mechanism stays dormant (further evading detection during black box testing), only activating when a preset condition is matched (this happens with a very low probability unless the mechanism is known). One common way to do this is with a *Hardware Trojan* (HT) [4, 20], where the chip manufacturer implants a low footprint component on top of the designer-specified circuit. The presence of the HT and its access mechanism is known only to the chip manufacturer (neither the cipher designer nor the user knows about it), hence the manufacturer can trigger the HT at point thereby making the chip to leak sensitive information.

Our interest, however, lies on the other type of backdoor, which works at the cipher design level. In this case, the cipher is so designed that, it has a secret backdoor which is known only by its designers. It is long been speculated that some ciphers, whose entire specification is available in public, may contain some secret backdoor. It is possible that one (or more) such cipher will eventually become a part of an international standard and subsequently be adopted by industry and academia alike; while its designers will sneak on sensitive information processed by this cipher.

The main differences between these two types are noted here. First, not every cipher has the second vulnerability, but every cipher affected in the first type as the cipher itself is made irrelevant. Also, it is possible to implant a first type of backdoor on a cipher having a second type backdoor. Second, the second type obeys the Kerckhoffs's principle (the design and security claims are made public), but the first type relies on vulnerability through obscurity (the complete information on manufacturing the chip/implementation of the network is not made public or kept vague/incomplete).

Each of these two types requires specialised knowledge, expertise and access. The first type is quite well explored, but this seems not to be the case for the second type. It can be argued that the second type is harder to build, as the cipher description, together with a somewhat clear design rationale is to be provided.

2.2 Context

The cipher level backdoors can be theoretically divided into the following categories:

(1) A cipher so craftily designed that nobody is able to find the presence of a backdoor after few years of speculation and testing. So far it passes all the known methods of cryptanalysis.

[12] For instance, one may look at the "politically correct" backdoor: https://www.kb.cert.org/vuls/id/247371.

[13] This is noted in [19, Section 1]: "There are two categories of backdoors. The first one is the backdoor implemented in a security product at the protocol or key-management level, which is generally considered in practice."

It is not known whether there is actually a backdoor or this is just a myth. If/when this cipher is made a standard and adopted as a global standard, it is in theory possible for its designers to access the backdoor and retrieve sensitive information.

The SIMON and SPECK [3] family contain few block ciphers which are suspected to have this kind of backdoor (see, e.g., [19]). It is not known if there is any backdoor, or how the backdoor mechanism works; if there is any in the first place. If there is some backdoor in those ciphers, it is never accessed, to the best of our knowledge/understanding.

(2) A cipher where the designers publicly claim there is a backdoor. The cipher is secure except when the backdoor is accessed. The designers make no attempt to hide the backdoor; rather they claim upfront that there is a backdoor – this is how the backdoor mechanism works – so on and so forth.

This category is recently popularised through the Malicious framework [19]. This framework can create such a cipher by tinkering with some otherwise secure cipher, given the base cipher satisfies some criteria. By accessing the backdoor, the cipher designer can retrieve the key by analysing the cipher output (at least in theory). The LOWMC-M [19] is an instance of this framework, which takes a secure instance of LOWMC as the base cipher.

One may notice the following characterisation of this category:

(α) The presence of the backdoor is made public by the designers without anybody asking. This also nullifies the question of whether it is hard to spot the presence of the backdoor had it not been known[14].

(β) Except when the backdoor is accessed, the cipher is secure. When the backdoor is accessed, the secret key is released from the cipher output (assuming the user does not prevent that, see Inference (B)) – at least in theory, satisfying the "practicality" condition of [19, Section 2.2].

(γ) Though the backdoor mechanism is public, it is infeasible to find out what the secret backdoor entry that activates the mechanism is. This is called the "undiscoverability" condition [19, Section 2.2].

By observing the Category (2) design in the literature (namely, [19]), the following inferences can be drawn (see Section 3.1 for more discussion):

(A) The backdoor can be accessed at most once. The backdoor mechanism is public, therefore anyone can check the incoming requests to the user to see if the backdoor is activated. Once it is found, the secret backdoor access becomes visible to everyone. Basically, one can monitor all the incoming traffic to the user and attempt to reverse-engineer the backdoor entry, and will eventually succeed as soon as the backdoor entry is used.

This can be done by the user (Alice) or the attacker (Eve[15]) alike. Though Eve does not know the actual secret key (which is chosen and kept secret by Alice), she can still choose an arbitrary key, and then follow-through the steps of the backdoor mechanism to see if information about the arbitrarily chosen key is released.

(B) It is not possible to extract the secret key without the user's (Alice's) cooperation. Alice can always keep an eye out for activation of the backdoor mechanism. Based on that, she may return an invalid output or something random (if the backdoor mechanism is activated), instead of the actual output from the cipher.

[14] It may be hard to spot the backdoor for someone who does not know beforehand, but here it does not matter as the designers have already made it public.

[15] As per [19, Section 2.1], the attacker/eavesdropper Eve is considered within the Malicious framework.

(C) The key which is released from the cipher output (as a consequence of the backdoor access) is not encrypted[16], meaning pretty much everyone on the network (including the attacker) can access it.

As any cipher belonging to Category (1) is not known[17], the following questions remain unanswered:

(a) Is it possible to design a backdoor where the backdoor mechanism and/or backdoor entry can stay hidden with the cipher specification being public but the alleged backdoor being accessed never?
(b) Is it possible to design a backdoor where the backdoor mechanism and/or backdoor entry can stay hidden with the cipher specification being public and after the backdoor being accessed (at least) once?
(c) Is it possible to design a backdoor where the user cannot sense that the backdoor is being accessed (or equivalently, the backdoor mechanism does not become visible when accessed)?
(d) Is it possible to design a backdoor where the user can sense that the backdoor is being accessed, but cannot (efficiently) reverse-engineer the backdoor entry?
(e) Is it possible that the key released through backdoor access is somehow protected[18], so that the attacker does not get it (but the cipher designer can still get it)?
(f) Is it possible for the designer to retrieve the key only from one-way communication (either Alice $\rightarrow$ Bob or Bob $\rightarrow$ Alice)? Is it possible for the designer not to retrieve the secret key only from one-way communication, but from both-way communication (Alice $\leftrightarrow$ Bob)?

In this work, we aim at improving Category (2) backdoors; i.e., we are interested to create improved design that satisfies Criteria (2α), (2β) and (2γ). It is important to note that those criteria are adopted from [19], and are not conceived by us. It is perhaps worth noting that Inferences (A) and (B) violate the "untraceability" condition which is described in [19, Section 2.2] (Inference (B) is already acknowledged in [19, Section 5.3] as a violation of "untraceability"). Whether or not it is possible to design a Category (1) backdoor is left as an open problem.

Remark 1. The closest to Category (1) the designers of `Malicious` could have gone is to present a new cipher/framework/mode and make vague claim about presence/absence of a backdoor. Then it would be up to the community to figure out if there is a backdoor, how to activate the backdoor/how the backdoor mechanism works, etc.

Remark 2. In theory, it is possible to design a cipher in Category (1) if the designer manages to find an attack not yet known to the (mainstream) community[19]. The backdoor in this case will be activated through this new attack, and will (more than likely) be missed by the community

[16] If the released key is encrypted with another key, that means the cipher designer and the user have to know the other key beforehand. In that case, they can simply use any cipher (with the other key) to communicate the key released through the backdoor instead, thus completely cutting off the need for a backdoor.

[17] The closest example to Category (1) is probably the infamous `DUAL_EC_DBRG`, which is confirmed by Snowden in 2013. `DUAL_EC_DBRG` is supposed to be a secure CSPRNG. Unless re-seeded with an external source of entropy, its state can be recovered by observing an output, thus all subsequent outputs can be retrieved. Notice that, it does not exactly fit the basic requirement to a Category (1) cipher; as it cannot be initialised with a seed supplied by the designer, nor there is a secret key to recover.

[18] 'Protection' here includes, but not limited to, encryption. See Section 3.1 for relevant discussion.

[19] For instance, some of the public-key ciphers (including `RSA`) are now known to be vulnerable against quantum computers, but those attacks were not known when those ciphers were designed. In a less restricted sense, the quantum attacks can be considered as backdoors to those ciphers.

(at least until this attack is discovered or the backdoor mechanism is reverse-engineered). For perspective, the construction of `Malicious` [19] depends on the extremely popular differential attack (which is known for over three decades); thus the backdoor, in a very high likelihood, would be spotted by the community (had it not been known already).

3 Basic Concepts

3.1 Practical Application of a Backdoor

Status Quo The first problem that arises while talking about the practicality of backdoor is to convince the users to adopt it. There is no shortage of efficient ciphers in the public domain; with well-described design rationale and which are well-analysed by the community. The users, Alice and Bob, may simply refuse to adopt any new/experimental cipher (for example, any cipher from the `LOWMC` family [1] altogether, or the unusual choice of using an XOF to design an encryption as in `LOWMC-M` [19]), suspecting there could potentially be a backdoor. Therefore, in a loose sense, they agree for the designer to retrieve the secret key if they agree to adopt a new cipher. Thus, the design and study of backdoor appears to be purely an academic interest than a pragmatic one.

Anyway, as far the technical problems are concerned with the current concept of backdoor [19] (which we call Category (2), see Section 2.2), we note the following: Since the identity of the cipher designer (whom we call Derek for simplicity, as indicated in Section 1.2) is known to everybody in the network; including Alice (sender), Bob (recipient) and Eve (attacker). Therefore Alice (as well as Bob) can be extra cautious when a request comes from Derek, implying the limitations:

(i) Alice can simply deny any request from Derek, preventing him to access the backdoor.
(ii) If Alice complies with Derek's requests and lets him access the backdoor, this can be noticed by Eve. Now the secret key is leaked through the response from Alice and the key is not encrypted[20], thus Eve can effectively recover the key.

Overall, the Limitations (i) and (ii) mostly, if not fully, diminish any real-life application for a Category (2) backdoor. The cipher designer (Derek) cannot use it without active cooperation from the user (Alice or Bob). Even if Derek can obtain anonymous identity or spoof a fake identity, it is still up to the mercy of Alice. All the information is coming from Alice, so she can simply check the output from the cipher before sending it[21]; and discard the request or give a random output; should she suspect the backdoor is being accessed. On the other hand, if Alice agrees Derek to access the backdoor, they can instead create a secure channel between them (no need for a backdoor). Besides, letting Eve know the secret key is a miserable flaw, since the whole purpose of any cryptographic system is to ensure the attacker cannot access the key.

The point to note here is, we are heavily implying that the notion of backdoor, at least in its current form, suffers from severe limitation that comes from lack/absence of trust for Derek. If Alice does not respond to anyone she does not trust, anonymous/fake identity by Derek is

[20] There is practically no way to encrypt this key, at least within the realm of symmetric-key cryptography; as this would require exchange of another secret key between Alice and Derek. This invalidates the need for a backdoor in the first place.

[21] For instance, Alice can check if the XOR of two consecutive cipher outputs equals to the key. Given the backdoor mechanism is public, she already knows exactly what to look out for.

meaningless. We are not saying either of the assumptions is objectively true/untrue. We are simply saying, in order for Derek to succeed in utilising the backdoor; he needs to circumvent those real-life problems at first, which may turn out to be challenging.

The existing work [19] does not seem to address any such issue with adequate clarity[22], despite being published in a coveted venue. It leaves more questions unanswered than the number of questions it answers, in our humble opinion.

Uncertain Future Prospect While it does not seem possible to extract the secret key without cooperation from the user, it may be possible with some cipher in the future where the designer can extract the key in a way that the attacker cannot get it. One potential concept to achieve this in the future (that may or may not work) can be stated as follows.

Suppose, instead of only one backdoor entry, it is split into q backdoor shares[23] (somewhat comparable with the concept of secret-sharing [21]), where the cipher output from all the shares are required to retrieve the key. Say, by querying with the b_i backdoor entry, c_i is obtained, for $i = 0, \ldots, q-1$. Each c_i contains some information about the secret key, but all of those are required to get the key.

Not only that, each c_i is connected in secret way (which is only known to Derek) so that the connection is to be respected in order to find the key. With some suspension of disbelief, say, $k = \mathbf{f}(c_{j_0}, c_{j_1}, \ldots, c_{j_{q-1}})$ where the function $\mathbf{f}$ is secret (only known to Derek) and is not symmetric, for $(j_0, j_1, \ldots, j_{q-1})$ being secret a permutation of $(0, 1, \ldots, q-1)$. Thus, despite knowing all the public information as Derek does, Eve may not be able to actually uncover the key given certain regularity assumptions (like, q is sufficiently large) as she would need to cover the search-space of $q!$.

This concept is shared here only to pique the interest of the future researchers. Whether or not this will turn out to be a feasibility is unclear as of now.

3.2 Associated Notions of Security

Undetectability The authors of `Malicious` in of [19, Section 2.2] mention one desirable security notion for a Category (2) backdoor, "undetectability". It is defined as *"the inability for an external entity to realize the existence of the hidden backdoor"*. Here we argue that this is not a valid notion.

Note from Criterion (2α), the backdoor designers of `Malicious` [19] have already made the presence of the backdoor a public knowledge. Thus, it is a preconceived knowledge of the whole community that a backdoor exists, trivially violating "undetectability".

On a careful examination, we further notice that the notion about whether the cipher has an embedded backdoor does not seem to hold water either. This is because we are not aware of any possible way to ascertain a cipher does not contain a backdoor (*"How do you know* `AES` *does not have a backdoor?"*). The ciphers which are broken can be (arguably) considered to have a backdoor, but it does not seem possible to comment on non-existence of a backdoor about those ciphers which are deemed secure. As a consequence, it is not possible to say

[22] To be fair, the authors agree the "untraceability" condition is violated in their construction [19, Section 5.3]; still they do not analyse the extent of its consequence. Also, they seem to be completely oblivious to Eve's access to the secret key.

[23] Possibly something similar is laid out by Peyrin: https://thomaspeyrin.github.io/web/assets/docs/invited/TII_CRC_21_slides.pdf, Slide 63.

an arbitrary instance of LOWMC-M does not contain a backdoor (regardless of an intentional backdoor following Malicious is implanted or not), thereby making the analysis in [19, Section 5.1] useless.

We understand the sentiment why Peyrin and Wang [19] would want this notion of "undetectability". Just because we have a sentiment, it does not mean it is possible to achieve that in a rational/reasonable manner. In their case, what they actually do is to compare "a LOWMC instance with a Malicious-based backdoor" with "a LOWMC instance without a Malicious-based backdoor", and do some security evaluation. This does not conform to their definition, as they do not prove "the LOWMC instance without a Malicious-based backdoor" does not have any other kind of backdoor. For all we know and care, even though that instance does not have a Malicious-based backdoor, it is certainly possible for it to have some other kinds of backdoor which are not known yet. Thus, they actually compare between two instances of LOWMC; of which one surely has a backdoor, and the other can have a backdoor ($\neq$ surely does not have a backdoor). If one insists on having a notion similar to "undetectability", we recommend to make it specific with respect to a particular framework — something like, *"the inability for an external entity to distinguish between the two scenarios; where a particular framework is used to create a backdoor, versus where that particular framework is not used"*.

Need for White-box Security Given the analysis in Sections 2.2 and 3.1; it stands to reason that, Alice (as well as Bob) and Eve can reverse-engineer the backdoor mechanism as soon as the first query is made by Derek as long as the cipher specification is public. Indeed, no matter how the backdoor mechanism works, it has to trigger something (such as, some variable has to become 0, some loop has to terminate, and so on). If the cipher specification is known, then anybody can utilise such information no later than the correct backdoor entry is used.

Therefore, if we want the backdoor mechanism will not be revealed even after a backdoor entry is queried with, a basic condition is that the cipher specification is to be kept secret by Derek. However, this alone is not enough, since it is possible to reverse-engineer the cipher specification given its (unprotected) implementation (cf. the well-known cases of reverse-engineering RC4[24] or CRYPTO-1 in Mifare Classic RFID tag [16]). Thus, the implementations of the cipher (which are prepared and shared by Derek to Alice and Bob) practically have to be secure against the *white-box* [8, 9] attacks. In a white-box setting, the secret key is embedded in the cipher implementation in a way that it cannot be recovered. That said, one may notice the following differences from the usual white-box setting (cf. *obfuscation*[25]):

1. The cipher specification itself is secret in a backdoor setting, which is a more stringent requirement than usual white-box (where it is public).
2. The cipher designer supplies the implementations to the users, but he does not know the secret key. This contrasts the usual white-box setting where the implementer knows and embeds the key. It is not clear whether this is a more stringent requirement.

At this point, it is perhaps safe to assume, there is no proper real-life application of the concept of backdoor introduced in [19], at least in the mainstream academic community.

[24] https://web.archive.org/web/20010722163902/http://cypherpunks.venona.com/date/1994/09/msg00304.html.

[25] https://www.esat.kuleuven.be/cosic/blog/program-obfuscation/.

Somebody may still use a cipher like that if it is enforced[26]. For instance, assume the situation where there a push from the government to implant some intentional backdoor to compromise the security of products used by the common people. In that case, it is in theory possible to use a Category (2) cipher, with its full specification being available in public (and with no white-box protection). Our cipher ZUGZWANG can be in theory used in such a situation; but as academic researchers with a moral compass, we do not condone that. To the best of our finding, the only incident similar to this is rumoured in Australia back in 2018, but it seems to be officially denied[27].

4 ZUGZWANG: Constructing a Block Cipher with a Backdoor

One major observation from Malicious [19] is that, the only reason the user/attacker cannot retrieve the backdoor is the one-way property of the XOF. As it is known, a Feistel block cipher can use an OWF as its state-update (see, DES for an example), we adopt the idea to finally extend it to ZUGZWANG. Whether or not a similar construction is possible with SPN and ARX families, and whether some other idea is possible that does not involve any OWF, are left open for future research.

4.1 Fundamental Idea of ZUGZWANG

In its simplest form, ZUGZWANG is a 2-branch balanced Feistel network based block cipher that runs for n rounds (counting of rounds goes from 0 to $n-1$). It uses $f_i(K_i, c_L)$ as the round function for for the i^{th} round; where K_i is the corresponding round key, c_L is the plaintext or the intermediate ciphertext currently at the left branch. Each f_i has the property that it collapses if $c_L = \hat{p_0}$ (if i is even) or $c_L = \hat{p_1}$ (if i is odd), for some predefined $\hat{p_0}$ and $\hat{p_1}$. In this case, $\hat{p} = \hat{p_0}||\hat{p_1}$ constitutes the secret backdoor. Also note that, the last Feistel round does not have any swap operation between the two branches (so there are $n-1$ branch swaps).

Now, notice that, $\hat{p_0}$ and $\hat{p_1}$ cannot be used directly in the specification of f_i's (those cannot be passed as parameters of f_i's); otherwise Alice and Eve would trivially retrieve these. Thus, we run an OWF, $H(\cdot)$ first. This leads to pre-computing and storing $H(\hat{p_0})$ (respectively, $H(\hat{p_1})$) where i is even (respectively, odd) in the cipher specification as constants. Now that $H(\cdot)$ is used to $\hat{p_0}$ and $\hat{p_1}$, we need to apply it to c_L too. Ultimately, instead of directly checking whether $c_L = \hat{p_0}$ or $c_L = \hat{p_1}$, we are now checking if $H(c_L) = H(\hat{p_0})$ or $H(c_L) = H(\hat{p_1})$.

Although $H(\hat{p_0})$ and $H(\hat{p_1})$ are stored and accessible in the cipher specification, the following claims hold due to the property of $H(\cdot)$:

- **Pre-image resistance:** It is hard to retrieve $\hat{p_0}$ and $\hat{p_1}$.
- **Second pre-image resistance:** It is hard to find another $\hat{p_i}'$ ($\neq \hat{p_i}$) such that $H(\hat{p_i}') = H(\hat{p_i})$ for $i = 0, 1$.

Notice that the birthday-bound security for collision does not apply here. These claims constitute the secrecy of the backdoor entry given its H; i.e., no algorithm better than the brute-force search is known.

[26] One may compare with the government-issued (closed-source) applications to trace COVID-19 to some extent, though there is no separate recipient (Derek = Bob) and there is no secret key to recover.

[27] https://www.homeaffairs.gov.au/about-us/our-portfolios/national-security/lawful-access-telecommunications/myths-assistance-access-act.

Next, we need to use an encryption, $E_{K_i}(\cdot)$ inside f_i's to make the overall Feistel cipher secure. As some form of the check $H(c_L) = H(\hat{p_j})$ for $j = i \pmod 2$ has to be inside f_i, a natural choice is $f_i = E_{K_i}(H(c_L)) \oplus \alpha_i$ where $\alpha_i = E_{K_i}(H(\hat{p_j}))$. After n rounds of update are done, post-whitening keys (K_L, K_R) are XORed to the branches.

Notice that when $\hat{p_0}$ and $\hat{p_1}$ are fed to the left and right branches respectively, f_i's are always 0, and this holds irrespective of number of rounds. Basically, f_i (which is 0) is XORed to the right branch, making the content of the right branch going to the left branch in the next round. As the content of the right branch is what makes f_{i+1} collapse, the pattern continues. This allows us to create an arbitrary number of rounds, though for proper security in two directions (and with the assumption that E and H have desirable cryptographic properties) the number of rounds is required to be ≥ 4. Therefore, when fed with the secret backdoor $\hat{p} = \hat{p_0}||\hat{p_1}$, the ciphertext from the cipher does not depend on the Feistel round keys (K_i's). Rather it always follows the simple invariant relationship with the post-whitening keys (K_L, K_R), which leads to direct key recovery with 1 plaintext query as:

$$(c_L, c_R) = \begin{cases} (\hat{p_0} \oplus K_L, \hat{p_1} \oplus K_R) & \text{if number of Feistel rounds is odd,} \\ (\hat{p_1} \oplus K_L, \hat{p_0} \oplus K_R) & \text{if number of Feistel rounds is even.} \end{cases}$$

On the other hand, when a p ($\neq \hat{p}$) is used as the plaintext, the cipher works as secure Feistel block cipher. At each round, the state update can be compared to a Boolean derivative of E – it resembles a form of differential attack on E (but weaker since $H(\hat{p_0})$ and $H(\hat{p_0})$ constants). Given E is secure, any differential attack on E does not give any usable information. We thus conclude, 4 rounds of the `ZUGZWANG` construction can be considered to provide adequate security in two directions.

Extension to Other Symmetric-key Primitives It may be possible to extend the core idea of `ZUGZWANG` to other primitives in the symmetric-key cryptography (viz., stream cipher, hash function, MAC, AE and AEAD). However, it is not immediately apparent how such extension will pan out. For instance, it is possible to get a stream cipher from a block cipher by using a number of modes (e.g., `CTR`); but it is to be noted that the plaintext does not enter the state of a stream cipher per-se. As such, we may have to use a secret IV/nonce so that, (say) the key-stream becomes all-zero regardless of the secret key. This requires elaborate discussion, and hence is kept out-of-scope for this work.

Feistel Types, Branches and Rounds The basic idea can be generalised to more Feistel branches, wherein the secret backdoor entry is split into multiple branches. In that case, one also needs to decide on the type of the Feistel Network and the minimum number of rounds required. Analysis such such options is left open for future research.

Location of Whitening Keys/Backdoor on Decryption Note that, the key recovery through backdoor access in `ZUGZWANG` does not retrieve any Feistel round key, rather it retrieves the whitening keys (the post-whitening keys for encryption, to be more precise). If we take all the Feistel branches have a whitening key XOR (for maximum key recovery), then the question is whether to use pre- or post-whitening keys.

For simplification of notation, assume that we have a 128-bit and 2-branch `ZUGZWANG` construction with n-rounds. First, let us study the situation for encryption where the left

Table 1: Complexity of whitening key recovery in 128-bit ZUGZWANG construction

(a) 2-branch Feistel

Whitening		Backdoor Complexity	
#Pre	#Post	Encryption	Decryption
$0^{\ddagger}$	$2^{\ddagger}$	2^0	2^{128}
1	1	2^{64}	2^{64}
2	0	2^{128}	2^0
2	2	2^{128}	2^{128}

‡: Instantiated in Section 4.2

(b) 4-branch Feistel

Whitening		Backdoor Complexity	
#Pre	#Post	Encryption	Decryption
0	4	2^0	2^{128}
1	3	2^{32}	2^{96}
2	2	2^{64}	2^{64}
3	1	2^{96}	2^{32}
4	0	2^{128}	2^0
4	4	2^{128}	2^{128}

branch has a pre-whitening key (K'), the right branch does not have a pre-whitening key. In this case, $\hat{p_0}$ does not make the f_i's collapse for even rounds, but $\hat{p_0} \oplus K'$ does. Since the designer does not know which K', he has to brute-force over 64-bits. Therefore, he has to query with $\hat{p_0} \oplus K' || \hat{p_1}$ for all possible 2^{64} choices of K'. The correct guess of K' can be identified by the output at the end (which will depend on presence/absence of whitening keys and if n is even/odd).

Reflecting on this, we observe that the construction with post-whitening keys helps in backdoor access in encryption, but not in decryption. Similarly, pre-whitening helps in decryption, but not in encryption. The invariant property that the product of the complexities for the whitening key recovery at both sides remains the same as the brute-force search. This is an inherent property of the ZUGZWANG construction. As shown in Table 1, this cannot be improved by increasing the number of Feistel branches. Improving the whitening key recovery complexity from two sides can be considered as a future work. Further, as indicated in Table 1, if both the pre- and post-whitening keys are used; this particular backdoor mechanism ceases to exist.

To the best of our finding, no claim about the backdoor access from Bob's side is available in Malicious [19]. Thus, the notion of "practicability", which is introduced in [19, Section 2.2], is unclear for Malicious/LOWMC-M decryption.

4.2 A Concrete Instance of ZUGZWANG (Using AES and SHAKE)

We show an instance of ZUGZWANG[28] that uses a 2-branch balanced Feistel structure, 128-bit state, runs for 4 Feistel rounds, and uses 0 pre-whitening and 2 post-whitening keys. We choose AES-128 for encryption (E), and SHAKE-128 as XOF (H). See Algorithm 1 for its formal description, and Section C for some test cases.

The basic construction for ZUGZWANG is as shown in Figure 2. The 128-bit master key K and the 128-bit backdoor entry $\hat{p}$ are split into two 64-bit post-whitening keys: $K = K_4 || K_5$, $\hat{p} = \hat{p_0} || \hat{p_1}$ respectively. The 128-bit Feistel round keys (K_i for $i = 0, 1, 2, 3$) are generated by running AES in CTR mode with key k (i.e., with i as the plaintext).

As per the construction, the $H(\cdot)$ of $\hat{p_0}$ and $\hat{p_1}$ are computed and stored. Since these are to be used as the plaintext for AES-128 (Line 10), we take 128-bit output for these. Similarly, the plaintext/intermediate ciphertexts are to be used in AES as plaintexts (Line 11), so the outputs

[28] We use the same term, 'ZUGZWANG', to indicate the overall construction idea as well as the concrete instance.

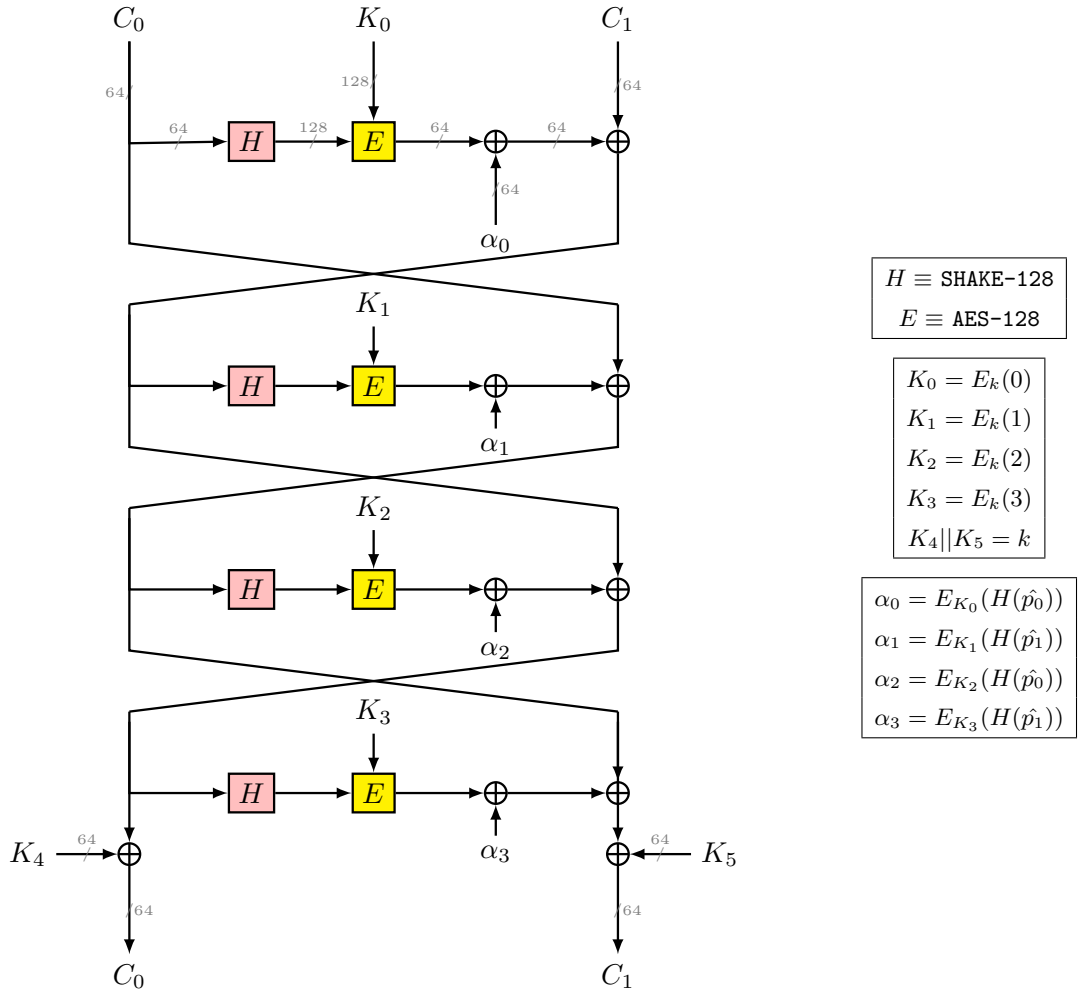

Fig. 2: `ZUGZWANG` (concrete instance/encryption)

from `SHAKE` for these are also taken as 128 bits long. However, since each Feistel branch is 64 bits long (see Line 12), we truncate the last 64 bits of these 128-bit `SHAKE` outputs.

If $\hat{p}$ is not known, then we claim this concrete instance offers 128-bit security (see Appendix B for a formal security proof). On the other hand, if queried with $\hat{p}$ as the plaintext, then the post-whitening keys (K_4, K_5) are revealed.

Remark 3. Instead of truncating the last 64 bits (Lines 10 and 11), some other methods (like, taking only the bits even/odd positions of the state, XORing of first 64 and last 64 bits) to get 64 bits from 128 bits can be used.

Remark 4. The Feistel round keys can be generated by some other means also, so long as the conditions for Feistel compatibility (see, e.g., [15]) are satisfied.

Remark 5. If some other cipher that supports 64-bit plaintext is used instead of `AES` (Lines 10 and 11), then truncation can be skipped.

Algorithm 1: ZUGZWANG – concrete instance/encryption (128 bits, 2-branch Feistel, 4 rounds; AES-128 as encryption, SHAKE-128 as XOF)

Input: $\hat{p}$ (backdoor entry, 128 bits), p (plaintext, 128 bits), k (master key, 128 bits)
Output: Secure encryption with key k and plaintext p if $p \neq \hat{p}$; k if $p = \hat{p}$

$n \leftarrow 4$ ▷ Number of Feistel rounds
for $i \leftarrow 0$ to $n-1$ **do**
  $K_i \leftarrow \texttt{AES}_k(i)$ ▷ Generate Feistel round keys by using AES in CTR mode
$K_4, K_5 \leftarrow k[0:63], k[64:127]$ ▷ Split k to use as post-whitening keys
$\hat{p_0}, \hat{p_1} \leftarrow \hat{p}[0:63], \hat{p}[64:127]$ ▷ Split $\hat{p}$ into 2 parts
Pre-compute and store $\texttt{SHAKE}(\hat{p_0})$, $\texttt{SHAKE}(\hat{p_1})$ ▷ Both are of 128-bits
$C_0, C_1 \leftarrow p[0:63], p[64:127]$ ▷ Split p into 2 parts
for $i \leftarrow 0$ to $n-1$ **do** ▷ Iterate over Feistel rounds
  $j \leftarrow i \pmod 2$
  $\alpha_i \leftarrow \texttt{AES}_{K_i}(\texttt{SHAKE}(\hat{p_j}))[0:63]$
  $\beta_i \leftarrow \texttt{AES}_{K_i}(\texttt{SHAKE}(C_0))[0:63]$ ▷ $\texttt{SHAKE}(C_0)$ is of 128-bits
  $f_i \leftarrow \beta_i \oplus \alpha_i$ ▷ $f_i = 0$ when $C_0 = \hat{p_j}$
  $C_1 \leftarrow C_1 \oplus f_i$ ▷ Update right Feistel branch with f_i
  if $i \leq n-2$ **then** ▷ No branch swap in last Feistel round
    $C_0, C_1 \leftarrow C_1, C_0$ ▷ Swap two Feistel branches
$C_0, C_1 \leftarrow C_0 \oplus K_4, C_1 \oplus K_5$ ▷ XOR post-whitening keys
return $C_0 || C_1$ ▷ $(C_0, C_1) = (\hat{p_1} \oplus K_4, \hat{p_0} \oplus K_5)$ when $p = \hat{p}$

ZUGZWANG is not meant to be used in practice, rather its primary function is to work as a proof-of-concept. Thus, we acknowledge the device footprint for the concrete instance can be significantly lowered (say, using less number of rounds for AES, replacing AES with a lightweight encryption, using an LFSR to generate Feistel round keys, etc.) but do not make any attempt to do so. For the same reason, we do not present any benchmark.

4.3 Comparison of ZUGZWANG with Malicious/LOWMC-M

In essence, the fundamental concept in Malicious [19] can be as described as follows. The backdoor entry is accessed through a (secret) difference at the (public) tweak and the (public) plaintext. The backdoor access works by cancelling the differences with one another in such a way that ultimately there is a high probability differential trail at the end, which potentially leaks the key. However, if just this much would be implemented, the attacker/user would (likely) notice the differences applied through the tweak and the plaintext, which would in turn reveal the backdoor entry. To prevent the attacker/user from obtaining the differences, the designers of Malicious [19] pass the differences through an XOF.

The following comparative points can be noted:

1. Malicious is based on somewhat new/ad-hoc and not so much analysed design principles. It cannot be implemented atop any pre-existing cipher, rather only applies to a cipher with meta-description given the meta-description satisfies certain requirement (e.g., a partially non-linear layer). Effectively, Malicious is too much reliant on the security of LOWMC [1]; any new cryptanalysis on LOWMC can potentially follow-through. Besides, being a new design type, itself requires its own analysis (in particular, LOWMC-M is revised in [19, Section 4.3] after a new analysis is presented in [12] where 7 instances of LOWMC-M with original parameters are broken without finding backdoor by algebraic attacks on LOWMC). ZUGZWANG can be designed

using already well-analysed primitives — all the concepts used in its construction/analysis are known for decades. Its security can be formally proven (Section B).

2. `Malicious` requires tweak (see Section A.1 for a possible extension that does not need a tweak). Tweak is said to be relatively new, less efficient, and any standard does not appear to exist [2, Section II.B]. `ZUGZWANG` does not require a tweak (in theory, a tweakable version can be constructed if needed).
3. The key recovery using the secret backdoor entry in `ZUGZWANG` is deterministic in nature, which requires only one call to the cipher with the secret backdoor entry equated with the plaintext; whereas `LOWMC-M` requires multiple calls[29], and the key recovery is not guaranteed even if its queries satisfy all the requisite conditions.
4. Since `LOWMC` exploits the differential property, the search space for the secret backdoor entry is actually less by 1 from the brute-force search (i.e., $2^{128} - 1$ for a 128-bit cipher). `ZUGZWANG` does not rely on differential, and the complexity for the backdoor access recovery is the same as the brute-force search.
5. The overall idea of `ZUGZWANG` is much easier to visualise, analyse and implement. When compared to the over-the-top design of `Malicious`/`LOWMC-M`, `ZUGZWANG` looks and feels more like an open-book.

5 Conclusion

Taking inspiration from Peyrin and Wang's Crypto'20 paper [19], we partake in a deeper dive at backdoor construction and related security concerns. Our analysis reveals a plethora of pertinent ideas/problems relating to [19]. A major contribution in our work is to present a block cipher concept, `ZUGZWANG`, that has an internal backdoor it. We also show a concrete instance of the concept. Our construction answers some of the open problems of `Malicious`/`LOWMC-M` [19], thus considerably improving from it.

Potential problems for future research are scattered throughout the paper, some of which are listed here:

- Is it possible to design a cipher with an innate backdoor (the cipher is secure otherwise) without using any OWF?
- How closely white-box cryptography and backdoor are related?
- Is it possible to 'hide' the key (which is released as the result of backdoor access) so that designer can get it but the attacker cannot?
- Is it possible to design a backdoor such that the complexity for the key recovery from both Alice's side and Bob's side are possible with trivial complexity?

In addition, we pose the theoretical problems for the follow-up works:

- Extend the concept for the public-key ciphers. One possible way could be so that Derek knows a secret plaintext ($\hat{p}$) so that Alice's secret key (k_s) can be computed given her public key (k_p); i.e., k_s is a computationally feasible function of $(\hat{p}, k_p)$.
- Extend the provable security claim of `ZUGZWANG` (e.g., more Feistel rounds, more Feistel branches).

To make ourselves clear, we do not support the government or any organisation forcing/tricking anybody to use any cipher that has a backdoor. We believe the intentional design

[29] There is no clear estimate on how many queries would be required.

a cipher with a hidden backdoor should be done as an academic curiosity (and not for any practical application).

A Postscript

A.1 `Malicious` with a Regular Block Cipher

There is a known construction of a (variable length) block cipher from a tweakable block cipher [7]. If we use this construction from a tweakable block cipher with a backdoor, then it is in theory possible to have a block cipher that does not necessarily need any tweak. This makes the construction in [19] more general, and appears to be a missed opportunity.

A.2 A Known Backdoor in `Davies-Meyer` Mode

It is well-known that the `Davies-Meyer` mode contains a fixed point. Interestingly, this can be extended for a hash function such that two pre-images have the same hash output. This acts as a second pre-image attack, and can be considered a backdoor for a hash function.

A.3 Backdoor with a Public-key System

It seems possible to design a backdoor using a public key system that operates as a mode. For a quick illustration of the concept, consider `RSA`. The composite number n (where $n = pq$; p, q are primes) is stored beforehand. The secret key is released if[30] the plaintext m divides n (thus, $m =$ either p or q). Otherwise, some secure cipher, say `AES` (with a suitable mode/padding), is run with m as the plaintext.

Breaking this, in general, seems at least as hard as breaking `RSA`. Effectively, p and q independently act two backdoors. Also note that, p and q are to be larger than 128 bits to ensure adequate security.

A.4 Note on White-box Cryptography

Since we are on the topic of white-box cryptography (Section 3.2), we take a brief moment to look at the "theoretically secure" white-box construction. As per the existing literature (see, e.g., [14, Chapter 3.1.1]), it is theoretically possible to achieve white-box security by using a gigantic memory.

If we consider `AES-128` to simplify the notation; then based on this hypothesis, we could have had a theoretically secure white-box if we had a gigantic memory that could store 2^{128} (plaintext, ciphertext) pairs for a fixed key. Since all of the responses to the plaintext queries are pre-computed and stored, therefore the key will not be used at all in the device, making it impossible for the attacker to retrieve the key. The proponents of this hypothesis point out that this far beyond what today's technology is capable of, and this is why we do not have a secure white-box system.

As we uncover here, the assumption of a gigantic memory does not ensure white-box security; such a construction – if/when exists – will be broken instantly. In other words, "if

[30] An example of the 'if–else' check in a cipher construction can be found in verification of the tag at the recipient's side for an AEAD.

only we had a gigantic memory…" argument displays the tip of the iceberg and hides the real problem underneath. Imagine, we have a gigantic memory, but it will not fill itself with all the (plaintext, ciphertext) pairs. For that what we need is a super powerful processor which can presumably do the followings:

(A) Generate all 2^{128} plaintexts.
(B) Encrypt all 2^{128} plaintexts.
(Γ) Encode all the 2^{128} ciphertexts, compute the address of the memory location for all 2^{128} ciphertexts, communicate all the 2^{128} ciphertexts through the peripheral to the memory.

Even if we ignore Presumption (Γ), the processor is assumed to be powerful enough to brute-force the 128-bit key (by Presumptions (A) and (B)) already. Therefore, if such a white-box system is ever constructed, it will be immediately broken — by the same processor which has filled it up[31]! As a result, it remains unclear whether or not a theoretically sound white-box system can be constructed.

A.5 Note of LOWMC

As a food-for-thought, we make the open-ended proposal to call LOWMC [1] a *meta-cipher* henceforth (instead of a cipher as it is commonly referred to). One may note that, LOWMC does not have fully defined specification, rather it has a meta description of the key schedule and the linear layer.

In a typical symmetric-key cipher, no component save for the secret key, is determined at random. Typically, the public-key ciphers (including the quantum-secure ciphers) which need some input which are not the secret key to be set at random. However, in all situations, the full cipher specification is already available beforehand. The random inputs are used to the fully specified cipher to initialise some components; but not as inputs to the cipher specifier algorithm to generate the exact cipher specification. Since LOWMC takes a random number to generate its full specification (the random number is not an input to the fully specified cipher), it possibly makes more sense to place LOWMC into the category of meta-ciphers.

B Security Proof of ZUGZWANG

B.1 Notations

Given any sequence $X = X_1 \cdots X_x$ and $1 \leq a \leq b \leq x$, we represent the sub-sequence $X_a \ldots X_b$ by $X[a \cdots b]$. For integers $a \leq b$, we write $[a \cdots b]$ for the set $\{a, \ldots, b\}$, and for integers $1 \leq a$, we write $[a]$ for the set $\{1, \ldots, a\}$. We use $X_1, \cdots, X_n \xleftarrow{\$} \mathcal{X}$ to denote that we sample n elements $X_1, \cdots, X_n$ from the set $\mathcal{X}$ with replacement at random.

B.2 Preliminaries

Distinguishing Advantage For two oracles $\mathcal{O}_0$ and $\mathcal{O}_1$, an algorithm $\mathcal{A}$ which tries to distinguish between $\mathcal{O}_0$ and $\mathcal{O}_1$ is called a distinguishing adversary. $\mathcal{A}$ plays an interactive

[31] In fact, the problem is more fundamental than that. In order to build such a gigantic memory, one needs to run an EDA tool. A processor that is capable of running such an EDA tool is more than capable of brute-forcing the cipher.

game with $\mathcal{O}_b$ where b is unknown to $\mathcal{A}$, and then outputs a bit $b_{\mathcal{A}}$. The winning event is $[b_{\mathcal{A}} = b]$. The distinguishing advantage of $\mathcal{A}$ is defined as:

$$\mathbf{Adv}_{\mathcal{O}_1,\mathcal{O}_0}(\mathcal{A}) := |\Pr[b_{\mathcal{A}} = 1|b = 1] - \Pr[b_{\mathcal{A}} = 1|b = 0]| \,.$$

Let $\mathbf{A}[q,t]$ be the class of all distinguishing adversaries limited to q oracle queries and t computations. We define:

$$\mathbf{Adv}_{\mathcal{O}_1,\mathcal{O}_0}[q,t] := \max_{\mathbf{A}[q,t]} \mathbf{Adv}_{\mathcal{O}_1,\mathcal{O}_0}(\mathcal{A}) \,.$$

When the adversaries in $\mathbf{A}[q,t]$ are allowed to make both encryption queries and decryption queries to the oracle, this is written as $\mathbf{Adv}_{\pm\mathcal{O}_1,\pm\mathcal{O}_0}[q,q',t]$, where q is the maximum number of encryption queries allowed and q' is the maximum number of decryption queries allowed. Enc_b and Dec_b denote respectively the encryption and decryption function associated with $\mathcal{O}_b$.

$\mathcal{O}_0$ conventionally represents an ideal primitive, while $\mathcal{O}_1$ represents either an actual construction or a mode of operation built of some other ideal primitives. Typically the goal of the function represented by $\mathcal{O}_1$ is to emulate the ideal primitive represented by $\mathcal{O}_0$. We use the standard terms real oracle and ideal oracle for $\mathcal{O}_1$ and $\mathcal{O}_0$ respectively. A security game is a distinguishing game with an optional set of additional restrictions, chosen to reflect the desired security goal. When we talk of distinguishing advantage with a specific security game $\mathcal{G}$ in mind, we include $\mathcal{G}$ in the superscript, e.g., $\mathbf{Adv}^{\mathcal{G}}_{\mathcal{O}_1,\mathcal{O}_0}(\mathcal{A})$. Also we sometimes drop the ideal oracle and simply write $\mathbf{Adv}^{\mathcal{G}}_{\mathcal{O}_1}(\mathcal{A})$ when the definition of the ideal oracle is clear from the context.

Coefficients H Technique The H-coefficient technique is a proof method by Patarin [17] that is modernised by Chen and Steinberger [5,6]. An adversary $\mathcal{A}$ interacts with oracles $\mathcal{O}$ (The oracle $\mathcal{O}$ could be a sequence of multiple oracles.) and obtains outputs from a real world $\mathcal{O}_1$ or an ideal world $\mathcal{O}_0$. The results of its interaction are collected in a transcript τ. The oracles can sample random coins before the experiment (often a key or an ideal primitive that is sampled beforehand) and are then deterministic. A transcript τ is attainable if $\mathcal{A}$ can observe τ with non-zero probability in the ideal world. The fundamental theorem of the Coefficients H Technique (proof of which can be found, e.g., in [5,6,17]) is stated in Theorem 1.

Theorem 1 (Patarin [17]). Assume, there exist $\epsilon_1, \epsilon_2 \geq 0$ such that $\Pr_{\mathcal{O}_0}[\mathsf{Bad}] \leq \epsilon_1$, and for any attainable transcript τ obtained without encountering Bad, $\frac{\Pr_{\mathcal{O}_1}[\tau]}{\Pr_{\mathcal{O}_0}[\tau]} \geq 1 - \epsilon_2$. Then, for all adversaries $\mathcal{A}$, it holds that: $\mathbf{Adv}_{\mathcal{O}_0,\mathcal{O}_1}(\mathcal{A}) \leq \epsilon_1 + \epsilon_2$.

The technique has been generalised by Hoang and Tessaro [10] in their expectation method, which allows them to derive the fundamental theorem as a corollary.

B.3 Design Rationale

Before we dive into the security proof of ZUGZWANG, we briefly justify our choice of number of rounds of the underlying Luby-Rackoff construction [13]. We know that at least three Feistel rounds are required to prove any sort of PRP (Pseudo Random Permutation) security (birthday-bound PRP security in case of three Feistel rounds, to be exact) [13], and at least four

Feistel rounds are required to prove any sort of SPRP (Strong Pseudo Random Permutation) security (birthday-bound SPRP security in case of four Feistel rounds, to be exact) [11,13,18] of the Luby-Rackoff construction. As we intend to show that ZUGZWANG behaves as an SPRP up to the allowed query complexity of the adversary, we require at least four Feistel rounds. To keep things as simple as possible, we have opted for that minimum number of Feistel rounds, which is four.

B.4 Distinguishing Game

As we will prove that the SPRP (Strong Pseudo Random Permutation) security of our construction using Coefficients H technique in standard model, at the start of the game, we sample four permutations $p_0, p_1, p_2, p_3 \xleftarrow{\$} \mathsf{Perm}$, where Perm is the set of all permutations ever $\{0,1\}^n$, and replace the four block ciphers in our construction with them. To prove the distinguishing advantage of the adversary, we use the standard hybrid argument, and the first component of the distinguishing advantage will be the PRP (Pseudo Random Permutation) advantage of the block cipher (the maximum advantage of any adversary to distinguish a PRP from the block cipher, when instantiated by a key which is chosen from $\{0,1\}^n$ uniformly).

The adversary ($\mathcal{A}$) makes q queries (forward or backward) to the oracle ($\mathcal{O}_1$, the real oracle; or $\mathcal{O}_0$, the ideal oracle). We call the i^{th} query to the oracle as $P_0^i \| P_1^i$ if it is a forward query, and the corresponding response as $C_0^i \| C_1^i$. Similarly, we call the i^{th} query to the oracle as $C_0^i \| C_1^i$ if it is a backward query, and the corresponding response as $P_0^i \| P_1^i$. $\mathcal{A}$ also makes q' queries to the hash ($\mathcal{H}$). We call the i^{th} hash query as U^i and the corresponding response as V^i. We call this as the online phase of the game. Note that $\mathcal{H}$ is same in both the real world and the ideal world.

We assume that $\mathcal{A}$ is non-repeating, i.e., it follows the following two properties while making queries to the oracle or the hash.

1. It never queries $P_0^i \| P_1^i$ or $C_0^i \| C_1^i$ ($i \in [q]$) to the oracle if it has already queried $P_0^i \| P_1^i$ or $C_0^i \| C_1^i$ (and received $C_0^i \| C_1^i$ or $P_0^i \| P_1^i$ in response respectively).
2. It never queries U^i ($i \in [q']$) to the hash if it has already queried it.

Once $\mathcal{A}$ is done with all its queries, the oracle releases some extra information to $\mathcal{A}$ which it can take into account before submitting its decision. We call this as the offline phase of the game. While the real oracle releases all the corresponding true values throughout this game, the ideal oracle releases them according to the following sampling method. We shall go on mentioning the bad events simultaneously, as soon as they can possibly occur. Note that whenever we mention any bad event or reach any sampling step, it is implicitly mentioned that the previously mentioned bad events have not occurred until that point.

B.5 Sampling of Ideal Oracle and Bad Events

Even before the start of the online phase, we define two sets Dom and Ran and initialize them with $\emptyset$ (i.e., null set). During the online phase, whenever $\mathcal{A}$ makes any hash query, we update Dom and Ran with the input and the output respectively. In our discussion, updating a set with an element means adding that element to that set if it is not already present in the set. Note that the variables in each bad event which are the sources of randomness in the context

of bounding the probability of that bad event in Section B.6 are marked in brown. The secret backdoor entry is denoted by the variable, $\hat{P}$, where $\hat{P} = \hat{P}_0||\hat{P}_1$, and $\hat{P}_i$ is chosen at random from $\{0,1\}^n$ (where n = size of one Feistel branch = 64).

Online Phase

1. For all $i \in [q]$, $C_0^i, C_1^i \xleftarrow{\$} \{0,1\}^n$, if it is a forward oracle query. Otherwise, $P_0^i, P_1^i \xleftarrow{\$} \{0,1\}^n$, if it is a backward oracle query.

Bad1 $\exists i, j \in [q]$ with $i < j$ such that $C_0^i = C_0^j \vee C_1^i = C_1^j$, when the j-th query is a forward query, and $P_0^i = P_0^j \vee P_1^i = P_1^j$, when the j-th query is a backward query.

Bad2 $\exists i, j \in [q']$ with $i < j$ such that $V^i = V^j$.

Offline Phase

1. $K_4, K_5, \hat{P}_0, \hat{P}_1 \xleftarrow{\$} \{0,1\}^n$.

Bad3 $\exists i, j \in [q]$ such that $P_0^i = C_0^j + K_4$.

Bad4 $\exists i \in [q']$ such that $U^i = \hat{P}_0 \vee U^i = \hat{P}_1$.

Bad5 $\exists i \in [q]$ such that $P_0^i = \hat{P}_0 \vee P_0^i = \hat{P}_1 \vee C_0^i + K_4 = \hat{P}_0 \vee C_0^i + K_4 = \hat{P}_1$.

2. $\mathcal{H}(\hat{P}_0), \mathcal{H}(\hat{P}_1) \xleftarrow{\$} \{0,1\}^n$.

Bad6 $\exists i \in [q']$ such that $V^i = \mathcal{H}(\hat{P}_0) \vee V^i = \mathcal{H}(\hat{P}_1)$.

3. Update Dom with $\hat{P}_0$ and $\hat{P}_1$ and Ran with $\mathcal{H}(\hat{P}_0)$ and $\mathcal{H}(\hat{P}_1)$.
4. For $P_0^i \neq P_0^j$ where $i \in [q]$ and $j \in [i-1]$, if $P_0^i \notin$ Dom, set $X^i \xleftarrow{\$} \{0,1\}^n$.
5. For $C_0^i \neq C_0^j$ where $i \in [q]$ and $j \in [i-1]$, if $C_0^i + K_4 \notin$ Dom, set $W^i \xleftarrow{\$} \{0,1\}^n$.

Bad7 $\exists i, j \in [q]$ with $i \neq j$ such that $P_0^i \neq P_0^j$ but $X^i = X^j$.

Bad8 $\exists i, j \in [q]$ with $i \neq j$ such that $C_0^i \neq C_0^j$ but $W^i = W^j$.

Bad9 $\exists i, j \in [q]$ such that $X^i = W^j$.

Bad10 $\exists i \in [q]$ and $j \in [q']$ such that $P_0^i \neq U^j$ but $X^i = V^j$.

Bad11 $\exists i \in [q]$ and $j \in [q']$ such that $C_0^i + K_4 \neq U^j$ but $W^i = V^j$.

Bad12 $\exists i \in [q]$ such that $X^i = \mathcal{H}(\hat{P}_0) \vee X^i = \mathcal{H}(\hat{P}_1) \vee W^i = \mathcal{H}(\hat{P}_0) \vee W^i = \mathcal{H}(\hat{P}_1)$.

6. Update Ran with freshly sampled X^i's and W^i's and Dom with corresponding P_0^i's and $(C_0^i + K_4)$'s.
7. For $X^i \neq X^j$ where $i \in [q]$ and $j \in [i-1]$, set $\hat{X}^i \xleftarrow{\$} \{0,1\}^n$.
8. For $W^i \neq W^j$ where $i \in [q]$ and $j \in [i-1]$, set $\hat{W}^i \xleftarrow{\$} \{0,1\}^n$.
9. $\alpha_0, \alpha_1, \alpha_2, \alpha_3 \xleftarrow{\$} \{0,1\}^n$.

Bad13 $\exists i, j \in [q]$ with $i < j$ such that $X^i \neq X^j$ but $\hat{X}^i = \hat{X}^j$.

Bad14 $\exists i, j \in [q]$ with $i < j$ such that $W^i \neq W^j$ but $\hat{W}^i = \hat{W}^j$.

Bad15 $\exists i \in [q]$ such that $\hat{X}^i = \alpha_0 \vee \hat{W}^i = \alpha_3$.

Bad16 $\exists i, j \in [q]$ with $i < j$ such that $A^i = A^j \vee B^i = B^j$, or in other words, $\hat{X}^i + \hat{X}^j = P_1^i + P_1^j$.

Bad17 $\exists i, j \in [q]$ such that $A^i = B^j$, or in other words, $P_1^i + \hat{X}^i + \alpha_0 = C_1^j + \hat{W}^j + \alpha_3 + K_5$.

Bad18 $\exists i \in [q]$ and $j \in [q']$ such that $A^i = U^j \vee B^i = U^j$, or in other words, $\hat{X}^i + P_1^i + \alpha_0 = U^j \vee \hat{W}^i + C_1^i + K_5 + \alpha_3 = U^j$.

Bad19 $\exists i \in [q]$ such that $A^i = \hat{P}_0 \vee A^i = \hat{P}_1 \vee B^i = \hat{P}_0 \vee B^i = \hat{P}_1$, or in other words, $\hat{X}^i + \alpha_0 + P_1^i = \hat{P}_0 \vee \hat{X}^i + \alpha_0 + P_1^i = \hat{P}_1 \vee \hat{W}^i + \alpha_3 + C_1^i + K_5 = \hat{P}_0 \vee \hat{W}^i + \alpha_3 + C_1^i + K_5 = \hat{P}_1$.

Bad20 $\exists i, j \in [q]$ such that $A^i = P_0^j \vee A^i = C_0^j + K_4 \vee B^i = P_0^j \vee B^i = C_0^j + K_4$, or in other words, $\hat{X}^i + \alpha_0 + P_1^i = P_0^j \vee \hat{X}^i + \alpha_0 + P_1^i = C_0^j + K_4 \vee \hat{W}^i + \alpha_3 + C_1^i + K_5 = P_0^j \vee \hat{W}^i + \alpha_3 + C_1^i + K_5 = C_0^j + K_4$.

10. For $i \in [q]$, set $Y^i \xleftarrow{\$} \{0,1\}^n$.
11. For $i \in [q]$, set $Z^i \xleftarrow{\$} \{0,1\}^n$.

Bad21 $\exists i, j \in [q]$ with $i < j$ such that $Y^i = Y^j \vee Z^i = Z^j$.

Bad22 $\exists i, j \in [q]$ such that $Y^i = Z^j$.

Bad23 $\exists i \in [q]$ and $j \in [q']$ such that $Y^i = V^j \vee Z^i = V^j$.

Bad24 $\exists i \in [q]$ such that $Y^i = \mathcal{H}(\hat{P}_0) \vee Y^i = \mathcal{H}(\hat{P}_1) \vee Z^i = \mathcal{H}(\hat{P}_0) \vee Z^i = \mathcal{H}(\hat{P}_1)$.

Bad25 $\exists i, j \in [q]$ such that $Y^i = X^j \vee Y^i = W^j \vee Z^i = X^j \vee Z^i = W^j$.

12. Update Ran with freshly sampled Y^i's and Z^i's and Dom with corresponding A^i's and B^i's.

Bad26 $\exists i, j \in [q]$ with $i < j$ such that $\hat{Y}^i = \hat{Y}^j \vee \hat{Z}^i = \hat{Z}^j$, or in other words, $P_0^i + P_0^j = C_1^i + C_1^j + \hat{W}^i + \hat{W}^j \vee C_0^i + C_0^j = P_1^i + P_1^j + \hat{X}^i + \hat{X}^j$.

Bad27 $\exists i \in [q]$ such that $\hat{Y}^i = \alpha_1 \vee \hat{Z}^i = \alpha_2$.

B.6 Probability Calculation of Bad Events

Table 2: Bad Events

Bad event	Range of indices	Upper bound
Bad1	$i, j \in [q], i < j$	q^2/N
Bad2	$i, j \in [q'], i < j$	q'^2/N
Bad3	$i, j \in [q]$	q^2/N
Bad4	$i \in [q']$	$2q'/N$
Bad5	$i \in [q]$	$4q/N$
Bad6	$i \in [q']$	$2q'/N$
Bad7	$i, j \in [q], i \neq j$	q^2/N
Bad8	$i, j \in [q], i \neq j$	q^2/N
Bad9	$i, j \in [q]$	q^2/N
Bad10	$i \in [q], j \in [q']$	qq'/N
Bad11	$i \in [q], j \in [q']$	qq'/N
Bad12	$i \in [q]$	$4q/N$
Bad13	$i, j \in [q], i < j$	q^2/N
Bad14	$i, j \in [q], i < j$	q^2/N
Bad15	$i \in [q]$	$2q/N$
Bad16	$i, j \in [q], i < j$	$2q^2/N$
Bad17	$i, j \in [q]$	q^2/N
Bad18	$i \in [q], j \in [q']$	qq'/N
Bad19	$i \in [q]$	$4q/N$
Bad20	$i, j \in [q]$	$4q^2/N$
Bad21	$i, j \in [q], i < j$	$2q^2/N$
Bad22	$i, j \in [q]$	q^2/N
Bad23	$i \in [q], j \in [q']$	$2qq'/N$
Bad24	$i \in [q]$	$4q/N$
Bad25	$i, j \in [q]$	$4q^2/N$
Bad26	$i, j \in [q], i < j$	$2q^2/N$
Bad27	$i \in [q]$	$2q/N$

In this part, we bound the probabilities of the bad events after fixing the values of the indices. Table 2 illustrates the range in which the indices varies, and bounds the probability for each bad event.

Bad1 If the j-th query is a forward query, then the probability of each of the events $C_0^i = C_0^j$ and $C_1^i = C_1^j$ comes out to be equal to $(1/N)$ due to the randomness of C_0^j and C_1^j respectively. Similarly, if the j-th query is a backward query, then the probability of each of the events $P_0^i = P_0^j$ and $P_1^i = P_1^j$ comes out to be equal to $(1/N)$ due to the randomness of P_0^j and P_1^j respectively.

Bad2 The probability of the event $V^i = V^j$ comes out to be equal to $(1/N)$ which is the collision security of the hash.

Bad3 The probability of the event $P_0^i = C_0^j + K_4$ comes out to be equal to $(1/N)$ due to the randomness of K_4.

Bad4 The probability of each of the events $U^i = \hat{P}_0$ and $U^i = \hat{P}_1$ comes out to be equal to $(1/N)$ due to the randomness of $\hat{P}_0$ and $\hat{P}_1$ respectively.

Bad5 The probability of each of the events $P_0^i = \hat{P}_0$ and $C_0^i + K_4 = \hat{P}_0$ comes out to be equal to $(1/N)$ due to the randomness of $\hat{P}_0$. Similarly, the probability of each of the events $P_0^i = \hat{P}_1$ and $C_0^i + K_4 = \hat{P}_1$ comes out to be equal to $(1/N)$ due to the randomness of $\hat{P}_1$.

Bad6 The probability of each of the events $V^i = \mathcal{H}(\hat{P}_0)$ and $V^i = \mathcal{H}(\hat{P}_1)$ comes out to be equal to $(1/N)$ due to the randomness of $\mathcal{H}(\hat{P}_0)$ and $\mathcal{H}(\hat{P}_1)$ respectively.

Bad7 If $P_0^i, P_0^j \in \mathsf{Dom}$, then the probability of the event $X^i = X^j$ comes out to be equal to 0 as the elements in Dom have distinct hash values. Without loss of generality, suppose $P_0^j \notin \mathsf{Dom}$. Then the probability of the event $X^i = X^j$ comes out to be equal to $(1/N)$ due to the randomness of X^j.

Bad8 If $C_0^i + K_4, C_0^j + K_4 \in \mathsf{Dom}$, then the probability of the event $W^i = W^j$ comes out to be equal to 0 as the elements in Dom have distinct hash values. Without loss of generality, suppose $C_0^j + K_4 \notin \mathsf{Dom}$. Then the probability of the event $W^i = W^j$ comes out to be equal to $(1/N)$ due to the randomness of W^j.

Bad9 If $P_0^i, C_0^j + K_4 \in \mathsf{Dom}$, then the probability of the event $X^i = W^j$ comes out to be equal to 0 as the elements in Dom has distinct hash values. If $P_0^i \notin \mathsf{Dom}$, then the probability of the event $X^i = W^j$ comes out to be equal to $(1/N)$ due to the randomness of X^i. Similarly, if $C_0^j + K_4 \notin \mathsf{Dom}$, then the probability of the event $X^i = W^j$ comes out to be equal to $(1/N)$ due to the randomness of W^j.

Bad10 The probability of the event $X^i = V^j$ comes out to be equal to $(1/N)$ due to the randomness of X^i.

Bad11 The probability of the event $W^i = V^j$ comes out to be equal to $(1/N)$ due to the randomness of W^i.

Bad12 The probability of each of the events $X^i = \mathcal{H}(\hat{P}_0)$ and $W^i = \mathcal{H}(\hat{P}_0)$ comes out to be equal to $(1/N)$ due to the randomness of $\mathcal{H}(\hat{P}_0)$. Similarly the probability of each of the events $X^i = \mathcal{H}(\hat{P}_1)$ and $W^i = \mathcal{H}(\hat{P}_1)$ comes out to be equal to $(1/N)$ due to the randomness of $\mathcal{H}(\hat{P}_1)$.

Bad13 The probability of the event $\hat{X}^i = \hat{X}^j$ comes out to be equal to $(1/N)$ due to the randomness of $\hat{X}^j$.

Bad14 The probability of the event $\hat{W}^i = \hat{W}^j$ comes out to be equal to $(1/N)$ due to the randomness of $\hat{W}^j$.

Bad15 The probability of each of the events $\hat{X}^i = \alpha_0$ and $\hat{W}^i = \alpha_3$ comes out to be equal to $(1/N)$ due to the randomness of α_0 and α_3 respectively.

Bad16 If the j-th query is a forward query and $P_0^i = P_0^j$ (which implies that $X^i = X^j$, from which it follows that $\hat{X}^i = \hat{X}^j$), then $P_1^i \neq P_1^j$ (because $\mathcal{A}$ is non-repeating) and the probability of the event $\hat{X}^i + \hat{X}^j = P_1^i + P_1^j$ comes out to be equal to 0. If the j-th query is a forward query and $P_0^i \neq P_0^j$, then the probability of the event $\hat{X}^i + \hat{X}^j = P_1^i + P_1^j$ comes out to be equal to $(1/N)$ due to the randomness of $\hat{X}^j$. If the j-th query is a backward query, then the probability of the event $\hat{X}^i + \hat{X}^j = P_1^i + P_1^j$ comes out to be equal to $(1/N)$ due to the randomness of P_1^j. Exactly similar analysis holds for the event $B^i = B^j$.

Bad17 The probability of the event $P_1^i + \hat{X}^i + \alpha_0 = C_1^j + \hat{W}^j + \alpha_3 + K_5$ comes out to be equal to $(1/N)$ due to the randomness of α_3.

Bad18 The probability of each of the events $\hat{X}^i + P_1^i + \alpha_0 = U^j$ and $\hat{W}^i + C_1^i + K_5 + \alpha_3 = U^j$ comes out to be equal to $(1/N)$ due to the randomness of α_0 and α_3 respectively.

Bad19 The probability of each of the events $\hat{X}^i + \alpha_0 + P_1^i = \hat{P}_0$ and $\hat{X}^i + \alpha_0 + P_1^i = \hat{P}_1$ comes out to be equal to $(1/N)$ due to the randomness of α_0. Similarly, the probability of each of the events $\hat{W}^i + \alpha_3 + C_1^i + K_5 = \hat{P}_0$ and $\hat{W}^i + \alpha_3 + C_1^i + K_5 = \hat{P}_1$ comes out to be equal to $(1/N)$ due to the randomness of α_3.

Bad20 The probability of each of the events $\hat{X}^i + \alpha_0 + P_1^i = P_0^j$ and $\hat{X}^i + \alpha_0 + P_1^i = C_0^j + K_4$ comes out to be equal to $(1/N)$ due to the randomness of α_0. Similarly, the probability of each of the events $\hat{W}^i + \alpha_3 + C_1^i + K_5 = P_0^j$ and $\hat{W}^i + \alpha_3 + C_1^i + K_5 = C_0^j + K_4$ comes out to be equal to $(1/N)$ due to the randomness of α_3.

Bad21 The probability of each of the events $Y^i = Y^j$ and $Z^i = Z^j$ comes out to be equal to $(1/N)$ due to the randomness of Y^j and Z^j respectively.

Bad22 The probability of the event $Y^i = Z^j$ comes out to be equal to $(1/N)$ due to the randomness of Z^j.

Bad23 The probability of each of the events $Y^i = V^j$ and $Z^i = V^j$ comes out to be equal to $(1/N)$ due to the randomness of Y^i and Z^i respectively.

Bad24 The probability of each of the events $Y^i = \mathcal{H}(\hat{P}_0)$ and $Y^i = \mathcal{H}(\hat{P}_1)$ comes out to be equal to $(1/N)$ due to the randomness of Y^i. Similarly, the probability of each of the events $Z^i = \mathcal{H}(\hat{P}_0)$ and $Z^i = \mathcal{H}(\hat{P}_1)$ comes out to be equal to $(1/N)$ due to the randomness of Z^i.

Bad25 The probability of each of the events $Y^i = X^j$ and $Y^i = W^j$ comes out to be equal to $(1/N)$ due to the randomness of Y^i. Similarly, the probability of each of the events $Z^i = X^j$ and $Z^i = W^j$ comes out to be equal to $(1/N)$ due to the randomness of Z^i.

Bad26 If the j-th query is a forward query, then the probability of each of the events $P_0^i + P_0^j = C_1^i + C_1^j + \hat{W}^i + \hat{W}^j$ and $C_0^i + C_0^j = P_1^i + P_1^j + \hat{X}^i + \hat{X}^j$ comes out to be equal to $(1/N)$ due to the randomness of $\hat{W}^j$ and C_0^j respectively. Similarly, if the j-th query is a backward query, then the probability of each of the events $P_0^i + P_0^j = C_1^i + C_1^j + \hat{W}^i + \hat{W}^j$ and $C_0^i + C_0^j = P_1^i + P_1^j + \hat{X}^i + \hat{X}^j$ comes out to be equal to $(1/N)$ due to the randomness of P_0^j and $\hat{X}^j$ respectively.

Bad27 The probability of each of the events $\hat{Y}^i = \alpha_1$ and $\hat{Z}^i = \alpha_2$ comes out to be equa to $(1/N)$ due to the randomness of α_1 and α_2 respectively.

B.7 Ratio of Good Interpolation Probabilities

Let the number of distinct hash calls to compute the first and fourth Feistel round output be q'_P and q'_C respectively. Then for any good transcript τ, we get,

$$\Pr_{\mathcal{O}_1}[\tau] = \left(\frac{1}{N^4}\right)\left(\frac{1}{(N)_{q'_P}(N)^2_q(N)_{q'_C}}\right)\left(\frac{1}{N^{|\mathsf{Dom}|}}\right).$$

Here we describe the terms in the denominator. The first term corresponds to the number of choices for the tuple $(K_4, K_5, \hat{P}_0, \hat{P}_1)$. The second and third terms correspond to the number of block cipher calls and the number of hash function calls respectively. We also get,

$$\Pr_{\mathcal{O}_0}[\tau] = \left(\frac{1}{N^4}\right)\left(\frac{1}{N^{2q}}\right)\left(\frac{1}{N^{q'_P+q'_C}}\right)\left(\frac{1}{N^{|\mathsf{Dom}|}}\right).$$

Here we describe the terms in the denominator. The first term corresponds to the number of choices for the tuple $(K_4, K_5, \hat{P}_0, \hat{P}_1)$. The second and third terms corresponds to the number of choices for the tuple $(P^i_0, P^i_1, C^i_0, C^i_1)$ and $(\hat{X}^i, \hat{W}^i)$ respectively where $i \in [q]$. The fourth term corresponds to the number of hash function calls. Thus we finally obtain, $\frac{\Pr_{\mathcal{O}_1}[\tau]}{\Pr_{\mathcal{O}_0}[\tau]} \geq 1$.

B.8 Distinguishing Advantage of Adversary

Suppose the PRP advantage of the block cipher for a key $K \xleftarrow{\$} \{0,1\}^n$ is $\mathbf{Adv}^{\mathrm{PRP}}_{E_K}(\mathcal{A}')$. Then we obtain from Sections B.6 and B.7, by applying the fundamental theorem of coefficients H technique and the standard hybrid argument:

$$\mathbf{Adv}_{\mathcal{O}_0,\mathcal{O}_1}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{PRP}}_{E_K}(\mathcal{A}') + \frac{23q^2 + q'^2 + 5qq' + 20q + 4q'}{N}.$$

The dominating terms in the right hand side apart from the PRP advantage of the block cipher are of the order of q^2/N, q'^2/N and qq'/N, which are all birthday-bound security terms; i.e., whenever the number of queries approaches $\sqrt{N}$, one of these terms approaches 1, depending on what type of query is being made; and as a result, the overall SPRP advantage of the adversary also approaches 1.

B.9 Case with Dependent Keys

The previous security proof considers the fact that the round keys are chosen independently and uniformly from $\{0,1\}^n$; and as a result, we are able to prove the birthday-bound security of `ZUGZWANG` in the standard model. But this condition is not necessary to prove the birthday-bound security of `ZUGZWANG`. For example, one can derive the round keys from the post-whitening keys using a proper key scheduling algorithm, and again prove the birthday-bound security of `ZUGZWANG`. The only difference in the later scenario is that we need to prove the security in the ideal cipher model. In fact, we have followed the second option while instantiating `ZUGZWANG`. The motivation behind our choice is that the second option makes the backdoor attack optimally powerful.

C Test Cases for ZUGZWANG (Concrete Instance)

In Table 3, we present some test cases for the concrete instance of ZUGZWANG (described in Section 4.2). The source-code (for encryption) can be accessed freely[32]. For perspective, the publicly available code[33] for Malicious apparently produces arbitrary-looking binary matrices only — it is not clear if/how the matrices can be used.

Table 3: ZUGZWANG (concrete instance) test cases

Key	10000000000000000000000000000000
Backdoor Entry	10000000000000000000000000000000
Plaintext	f0000000000000000000000000000000
Ciphertext	d5c03bdfec4e0fcf853645f83c6d74ef
Key	3cf28000471a74d22d81b6c98a3dbe33
Backdoor Entry	1831bcaa5bee08e639864e4823183090
Plaintext	1831bcaa5bee08e639864e4823183090
Ciphertext	0574ce486402444235b00a63d1d3b6d5
Recovered Key	3cf28000471a74d22d81b6c98a3dbe33
Key	3cf28000471a74d22d81b6c98a3dbe33
Backdoor Entry	1831bcaa5bee08e639864e4823183090
Plaintext	9eac455e039a58928e163658e1493a20
Ciphertext	7a7700e9f5f4f974e9ba2834160cf284

References

1. Albrecht, M., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for mpc and fhe. Cryptology ePrint Archive, Report 2016/687 (2016) https://eprint.iacr.org/2016/687. 2, 7, 14, 17
2. Baksi, A., Bhasin, S., Breier, J., Khairallah, M., Peyrin, T.: Protecting block ciphers against differential fault attacks without re-keying (extended version). Cryptology ePrint Archive, Report 2018/085 (2018) https://eprint.iacr.org/2018/085. 15
3. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404 (2013) https://eprint.iacr.org/2013/404. 1, 5
4. Bhasin, S., Danger, J.L., Guilley, S., Ngo, X.T., Sauvage, L.: Hardware trojan horses in cryptographic ip cores. Cryptology ePrint Archive, Paper 2014/750 (2014) https://eprint.iacr.org/2014/750. 4
5. Chen, S., Steinberger, J.: Tight security bounds for key-alternating ciphers. Cryptology ePrint Archive, Paper 2013/222 (2013) https://eprint.iacr.org/2013/222. 18
6. Chen, S., Steinberger, J.: Tight security bounds for key-alternating ciphers. In Nguyen, P.Q., Oswald, E., eds.: Advances in Cryptology – EUROCRYPT 2014, Berlin, Heidelberg, Springer Berlin Heidelberg (2014) 327–350 18
7. Chen, Y.L., Luykx, A., Mennink, B., Preneel, B.: Efficient length doubling from tweakable block ciphers. Cryptology ePrint Archive, Report 2017/841 (2017) http://eprint.iacr.org/2017/841. 16
8. Chow, S., Eisen, P., Johnson, H., Oorschot, P.C.V.: Oorschot. a white-box des implementation for drm applications. In: In Proceedings of ACM CCS-9 Workshop DRM, Springer (2002) 1–15 9
9. Chow, S., Eisen, P.A., Johnson, H., Oorschot, P.C.v.: White-box cryptography and an aes implementation. In: Revised Papers from the 9th Annual International Workshop on Selected Areas in Cryptography. SAC '02, Berlin, Heidelberg, Springer-Verlag (2002) 250–270 9
10. Hoang, V.T., Tessaro, S.: Key-alternating ciphers and key-length extension: Exact bounds and multi-user security. In Robshaw, M., Katz, J., eds.: Advances in Cryptology – CRYPTO 2016, Berlin, Heidelberg, Springer Berlin Heidelberg (2016) 3–32 18

[32] https://github.com/anubhab001/zugzwang-public.
[33] https://github.com/MaliciousLowmc/LowMC-M.

11. Hougaard, H.B.: 3-round feistel is not superpseudorandom over any group. IACR Cryptol. ePrint Arch. (2021) 675 19
12. Liu, F., Isobe, T., Meier, W.: Cryptanalysis of full lowmc and lowmc-m with algebraic techniques. Cryptology ePrint Archive, Paper 2020/1034 (2020) https://eprint.iacr.org/2020/1034. 14
13. Luby, M., Rackoff, C.: How to construct pseudorandom permutations from pseudorandom functions. SIAM J. Comput. **17**(2) (1988) 373–386 18, 19
14. Mulder, Y.D.: White-Box Cryptography, Analysis of White-Box AES Implementations. PhD thesis, Katholieke Universiteit Leuven (2014) https://core.ac.uk/download/pdf/34586143.pdf. 16
15. Nandi, M.: The characterization of luby-rackoff and its optimum single-key variants. In Gong, G., Gupta, K.C., eds.: Progress in Cryptology - INDOCRYPT 2010 - 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings. Volume 6498 of Lecture Notes in Computer Science., Springer (2010) 82–97 13
16. Nohl, K., Evans, D., Starbug, Plötz, H.: Reverse-engineering a cryptographic RFID tag. In van Oorschot, P.C., ed.: Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA, USENIX Association (2008) 185–194 9
17. Patarin, J.: The "Coefficients H" Technique. In Avanzi, R.M., Keliher, L., Sica, F., eds.: Selected Areas in Cryptography, Berlin, Heidelberg, Springer Berlin Heidelberg (2009) 328–345 18
18. Patel, S., Ramzan, Z., Sundaram, G.S.: Luby-rackoff ciphers: Why XOR is not so exclusive. In Nyberg, K., Heys, H.M., eds.: Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers. Volume 2595 of Lecture Notes in Computer Science., Springer (2002) 271–290 19
19. Peyrin, T., Wang, H.: The malicious framework: Embedding backdoors into tweakable block ciphers. Cryptology ePrint Archive, Report 2020/986 (2020) https://eprint.iacr.org/2020/986. 1, 2, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16
20. Ravi, P., Deb, S., Baksi, A., Chattopadhyay, A., Bhasin, S., Mendelson, A.: On threat of hardware trojan to post-quantum lattice-based schemes: A key recovery attack on SABER and beyond. In Batina, L., Picek, S., Mondal, M., eds.: Security, Privacy, and Applied Cryptography Engineering - 11th International Conference, SPACE 2021, Kolkata, India, December 10-13, 2021, Proceedings. Volume 13162 of Lecture Notes in Computer Science., Springer (2021) 81–103 4
21. Shamir, A.: How to share a secret. Commun. ACM **22** (1979) 612–613 https://dl.acm.org/doi/10.1145/359168.359176. 8