Counter Galois Onion: Fast Non-Malleable Onion Encryption for Tor

Jean Paul Degabriele[1], Alessandro Melloni[2], Jean-Pierre Münch[3], and Martijn Stam[2]

[1] Technology Innovation Institute, Masdar City, Abu Dhabi, U.A.E.
jeanpaul.degabriele@tii.ae
[2] Simula UiB, Bergen, Norway.
alessandro.melloni.29@gmail.com,martijn@simula.no
[3] Technische Universität Darmstadt, Darmstadt, Germany
jean-pierre.muench@posteo.de

Abstract. In 2012, the Tor project expressed the need to upgrade Tor's onion encryption scheme to protect against tagging attacks and thereby strengthen its end-to-end integrity protection. Tor proposal 261, where each encryption layer is processed by a strongly secure, yet relatively expensive tweakable wide-block cipher, is the only concrete candidate replacement to be backed by formal, yet partial, security proofs (Degabriele and Stam, EUROCRYPT 2018, and Rogaway and Zhang, PoPETS 2018).
We propose an alternative onion encryption scheme, called Counter Galois Onion (CGO), that follows a minimalistic, modular design and includes several improvements over proposal 261. CGO's underlying primitive is an updatable tweakable split-domain cipher accompanied with a new security notion, that augments the recently introduced rugged pseudorandom permutation (Degabriele and Karadžić, CRYPTO 2022). Thus, we relax the security compared to a tweakable wide-block cipher, allowing for more efficient designs. We suggest a concrete instantiation for the updatable tweakable split-domain cipher and report on our experiments comparing the performance of CGO with Tor's existing onion encryption scheme.
We identify the functionality and security desiderata for Tor's onion encryption, and show that our security notion for updatable tweakable split-domain ciphers successfully hybridizes, which allows us to argue informally that CGO meets the full security requirements.

Keywords: Tor · Onion Encryption · Tagging Attacks · Forward Security · RPRP

1 Introduction

Every day, the Tor network empowers millions of users by circumventing censorship and enabling anonymous Internet access. Since its inception, Tor has steadily increased in popularity, exhibiting multiple spikes in usage during censorship events around the globe [1]. Moreover, it has become an indispensable resource for whistleblowers, journalists, and activists to protect their identities while sharing sensitive information. The protocol at the heart of the Tor network evolved through a series of papers that developed onion routing [21,38,44,45], and it was eventually pinned down by Dingledine, Mathewson, and Syverson [16]. Since then, the circuit setup (key exchange) component has been replaced, and the Tor protocol has been augmented with numerous other extensions and protections.

Yet, the symmetric onion encryption responsible for protecting the data traffic over the Tor network, has remained unchanged, notwithstanding a clear need to replace it [33]: the current scheme's weaknesses include its malleability and susceptibility to tagging attacks, the rather short authentication tag, the fact that authentication tags are computed using a vulnerable MAC construction (albeit seemingly hard to exploit in Tor), and the fact that the onion encryption follows a MAC-then-encrypt approach—which in the case of authenticated encryption is known to be generically insecure.

In Tor, prior to sending any data, a sender first needs to establish a circuit typically consisting of three routers from the pool of available routers in the network. The sender would then share a symmetric key with every router in the circuit, and data traffic would flow through this circuit in encrypted form before it reaches its destination. Intuitively, anonymity is achieved because the full circuit is known only to the sender, whereas the routers in the network are only aware of the preceding and successive parties. The sender encrypts data as follows: it pads the message to a fixed size, appends a MAC computed over the full sequence of messages transmitted up to that point, and applies three layers of counter-mode encryption, a layer for every onion router in the circuit. As the ciphertext travels through the circuit, each router strips off one layer of encryption and forwards the resulting ciphertext to the next router in the circuit. The encryption layers serve to decorrelate the ciphertexts entering a router from those exiting it, assuming that the traffic flowing through the router is high enough so that incoming and outgoing ciphertexts cannot be easily correlated through timing.

Now, the sender's anonymity can be compromised if, for instance, an attacker can determine the first and last router in a circuit. In a tagging attack, an active adversary flips a bit in the ciphertext when it is travelling between the sender and the first router and then flips back this same bit just before the ciphertext reaches the last router. If decryption at the last router succeeds, the adversary has confirmed that the two routers are indeed on the same circuit. Note that it is the malleability of counter-mode encryption progressing through multiple layers that makes the scheme susceptible to tagging attacks. Interestingly, tagging attacks were already known and acknowledged by the Tor designers [16]. However they were deemed to be less damaging than passive traffic correlation attacks, which are unavoidable for a low-latency network like Tor. This opinion changed due to two anonymous posts on the Tor mailing list [36,37] that used the base-rate fallacy to argue that tagging attacks scale better than traffic correlation attacks and are thus more severe. This highlighted the need for better cryptography for Tor. The other issues about authentication, raised by Mathewson in Proposal 202 [33], refer to the computation of the authentication tag, which uses SHA1 by prepending the message with a secret key and truncating the tag to just 32 bits. This method is problematic both due to the relatively short tag and its use of a broken hash function in an insecure MAC construction, albeit seemingly unexploitable in Tor.

Designing an onion encryption scheme suitable for Tor presents several unique challenges. In addition to providing a forward secure channel between sender and receiver that is resistant against tagging attacks, the new onion encryption scheme should be able to coexist with the old scheme so as to permit a smooth transition. That is, it should be possible to establish heterogeneous circuits where some of the onion routers support the new scheme, and others do not. It should also support circuits of arbitrary length while maintaining a constant ciphertext size that is independent of both the circuit length and where in the circuit the ciphertext is processed. Another salient feature of the Tor protocol that must be supported by the onion encryption scheme is *leaky pipes*. This refers to the ability to send messages to any router within the circuit and not just the last (exit) router in the circuit. Accordingly, any suitable scheme must allow a router to determine unambiguously whether it is the intended recipient of a ciphertext, yet without leaking the intended recipient to any earlier routers on the circuit. Similarly, any router can send ciphertexts back to the sender, who in turn should be able to determine which router it originated from, yet intermediate routers should not learn the source router's identity. Finally, a significant challenge in replacing Tor's current onion encryption scheme is performance. The current scheme is rather simple in that it consists of a single keyed SHA1 evaluation for end-to-end integrity and AES in counter mode for each layer of encryption. In view of this minimalistic design, satisfying all the above requirements without incurring a substantial penalty in performance, is challenging.

In Proposal 202, besides expressing the need to update the onion encryption in Tor, Mathewson described two high-level ideas for a candidate replacement. The first was based on replacing each layer with a tweakable wide-block cipher and the latter can be described as a symmetric adaptation of Sphinx [11], a popular asymmetric scheme used in mix-nets. Eventually, the wide-block-cipher approach was deemed the more favourable one. Notably, it results in significantly less ciphertext expansion, with a ciphertext size independent of the circuit length, and it is functionally closer to Tor's current scheme, which facilitates retrofitting in Tor. Subsequently, Mathewson specified a concrete proposal for an onion encryption scheme based on the wide-block cipher approach [34], where AEZ was named as a potential candidate for instantiating the wide-block cipher [28]. The security of this scheme was analysed and proven secure in two concurrent and independent works [15, 41]. Shortly before, Ashur, Dunkelman, and Luykx [4] introduced GCM-RUP, a nonce-hiding AEAD scheme that is secure under the release of unverified plaintext (a.k.a. RUP security). They argued that replacing Tor's counter-mode encryption with a wide-block cipher would be overkill and proposed using GCM-RUP instead. However, there is a significant conceptual gap between an AEAD scheme and an onion encryption scheme, where transforming the former into the latter is not at all straightforward. Indeed, their associated onion encryption scheme [46] had no security proof, and it did not meet many of the desiderata we mentioned above.

1.1 Contribution

In this work we propose Counter Galois Onion (CGO)—a new onion encryption scheme for Tor that meets the requirements expressed in Proposal 202, provides forward security, and offers very competitive performance. Our scheme is inspired by the proposal of Ashur, Dunkelman, and Luykx [4,46], and serves to confirm their intuition that a wide-block cipher is not necessary to protect against tagging attacks.

However, as an onion encryption scheme, CGO improves significantly over theirs and our design rationale still differs considerable. Our contribution can be summarised as follows.

Modular and compact design. CGO is a concrete onion encryption scheme but its design follows a modular approach where the underlying building blocks can be easily replaced to improve security or performance if needed. In fact, we reveal the details of CGO in three stages using three different levels of abstraction. At the highest level of abstraction, CGO can be described in terms of a primitive that we call an updatable tweakable split-domain cipher, based on earlier work by Degabriele and Karadžić [13]. They introduced Rugged Pseudorandom Permutations (RPRP), essentially a security notion for split-domain tweakable ciphers that is weaker than the more common notion of strong tweakable pseudorandom permutations and can thus be attained by more lightweight constructions. A split-domain cipher is simply a cipher (family of permutations) operating over pairs of strings rather than strings. We augment these ciphers with an update mechanism for generating new key material, and introduce a corresponding security notion URRND, itself an extension of the RRND notion that Degabriele and Karadžić already mentioned as an alternative to RPRP. Intuitively, the update mechanism serves to make CGO stateful in order to provide forward security and meet other security requirements. We lay out the main rationale behind the design of CGO in Section 3.3.

At the second layer of abstraction, we describe how we instantiate the updatable split-domain cipher through the UIV+ construction, consisting of a tweakable blockcipher and a tweakable pseudorandom function. In principle, any split-domain cipher that is RPRP secure can be transformed into an updatable split-domain cipher that is URPRP secure by augmenting it with a separate pseudorandom function. Several RPRP-secure split-domain ciphers designs have been proposed [13, 14], all of which can thus be easily adapted for use in CGO via the simple transformation just described. However, the UIV+ construction employed in CGO realises the update functionality compactly without introducing additional components and key material by simply replacing the underlying pseudorandom function already present in UIV with a tweakable pseudorandom function. We then present a concrete instantiation of UIV+ that we call GCM-UIV+, where we realise the tweakable blockcipher and the tweakable pseudorandom function from AES and the universal hash function POLYVAL [22]. The main aim of GCM-UIV+ is to optimise CGO's performance by exploiting the native instruction sets commonly found in most modern CPUs allowing very fast implementations of AES and POLYVAL.

Performance benchmarks. Our final contribution is a performance analysis of CGO, benchmarked against Tor's existing onion encryption scheme, considering both the forward and backward directions. The simplicity of Tor's existing scheme makes it rather hard to outperform, especially at intermediate Onion Routers, where the cryptographic processing requires only the evaluation of counter-mode AES. At these nodes, we observe that CGO results in an overhead ranging between $20\% - 50\%$, where the higher overhead was observed in the backward direction. However, at the Onion Proxy and the Exit/Entry, we observe a speedup by a factor of three. This is where the SHA-1 processing takes place in Tor's existing scheme, which CGO avoids, leading to this more substantial improvement in performance. Arguably, the performance at the Onion Proxy and Exit/Entry node is more critical since Exit/Entry nodes are more scarce in the Tor network (due to their legal exposure), and the Onion Proxy could be running on a phone or some other lightweight device. Thus, when compared to Tor's existing onion encryption scheme, CGO provides superior security at the cost of a mild slowdown at the intermediate nodes and even better performance at the end nodes in the circuit. In view of this, we propose CGO as a practically viable replacement for Tor's existing onion encryption scheme that we believe meets all of the security and functional requirements originally outlined by Mathewson [33].

1.2 Related Works

Rugged pseudorandom permutations (RPRP) were introduced by Degabriele and Karadžić as a generic primitive that could be easily transformed into a variety of AEAD schemes with differing properties or compact Nonce-Set AEAD schemes from which order-resilient channels like QUIC and DTLS can be realised more easily [13]. A central focus of their work was to revisit the encode-then-encipher paradigm [8, 43] when instantiated with a weaker primitive, i.e., a rugged pseudorandom permutation instead of strong pseudorandom permutation. In contrast, in this work we consider the rather different task of building onion encryption from a rugged pseudorandom permutation. Our updatable rugged pseudorandom construction UIV+ is based on the UIV construction by Degabriele and Karadžić [13], which is itself based on the PIV construction by Shrimpton and Terashima [43]. In follow-up work, Degabriele and

Karadžić presented three further constructions of rugged pseudorandom permutations, which we believe could serve as the basis for alternative instantiations of CGO [14].

Ashur, Dunkelman, and Luykx introduced $\mathsf{GCM\text{-}RUP}$ [4] as an AEAD scheme which retains security under the release of unverified plaintext [2,6]. The authors suggested using $\mathsf{GCM\text{-}RUP}$ in Tor. While their initial idea was very insightful, turning a RUP-secure AEAD scheme into a suitable onion encryption scheme for Tor is non-trivial, as the latter needs to support leaky pipes and must be stateful in order to achieve chosen-plaintext security (otherwise the same message would always encrypt to the same ciphertext). They sketched out a concrete proposal how their AEAD scheme could be integrated into Tor to thwart tagging attacks [46], but still with several shortcomings as it did not support forward security and its non-modular structure made it particularly hard to analyse its security: while the AEAD scheme itself came with a security proof, the Tor proposal did not. Indeed, it did not protect against all types of tagging attacks, such as the ones described by Degabriele and Stam [15]. Our scheme CGO addresses all these shortcomings and additionally provides forward security.

Two other prior works [15,41] analysed the security of the scheme proposed by Mathewson in proposal 261 based on a strong pseudorandom permutation [34]. The two use rather different security models. Degabriele and Stam [15] formulate confidentiality, authenticity, and anonymity as separate security notions, and the routing mechanism, which affects security, is considered to be part of the scheme. On the other hand, Rogaway and Zhang [41] use an all-in-one security definition and focus solely on the onion encryption component rather than the complete onion routing scheme. Arguably, the approach adopted by Degabriele and Stam is more comprehensive and the security model is closer to the intuitive security goal, but it is also more complex, whereas the security model by Rogaway and Zhang is easier to work with. Unfortunately, neither of these prior works model leaky pipes or capture forward security, thus they are of limited use to analyse CGO.

2 Preliminaries

Notation. We use code-based experiments, where by convention all sets, lists, and lazy functions are initialized empty. Most security notions in this paper are captured by distinguishing advantages, where an adversary has to distinguish between the real experiment ($b = 1$) and some idealized version thereof ($b = 0$).

We use $\Pr[\mathit{Code} : \mathit{Event} \mid \mathit{Condition}]$ to denote the conditional probability of *Event* occuring when *Code* is executed, conditioned on *Condition*. We omit *Code* when it is clear from the context and *Condition* when it is not needed. In our (pseudo)code, we use the shorthand $\mathcal{X} \overset{\cup}{\longleftarrow} x$ for the operation $\mathcal{X} \leftarrow \mathcal{X} \cup \{x\}$, $\mathtt{X} \overset{\frown}{\longleftarrow} x$ to append the element x to the list $\mathtt{X}$, and the shorthand $(B, C) \twoheadleftarrow A$ to indicate that A is parsed as the tuple (B, C), where unique parsing should follow easily from the context. For any string x we denote by $\lfloor x \rfloor_\ell$ the substring consisting of the ℓ rightmost bits.

2.1 Tweakable Split-Domain Ciphers

A cipher is a family of length-preserving permutations over some domain $\mathcal{D}$, where each permutation is indexed by a key, or a key–tweak pair in the case of a tweakable cipher. We denote blockciphers and tweakable blockciphers by their respective enciphering algorithms,

$$\mathsf{E} : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n \quad \text{and} \quad \mathsf{E} : \{0,1\}^k \times \mathcal{H} \times \{0,1\}^n \to \{0,1\}^n ,$$

where k is the key size, n is the block size, and $\mathcal{H}$ is the tweak space. The conceptual building block behind our construction CGO will be yet another kind of cipher: a tweakable cipher over a *split domain*. Such ciphers were recently considered by Degabriele and Karadžić [13], who introduced *rugged pseudorandom permutations* (RPRP). An RPRP is a new security notion for ciphers that lies in between pseudorandom permutations [20] and strong pseudorandom permutations [32]. Intuitively, this intermediate security level is attained by providing the adversary full access to the enciphering algorithm but only partial access to the deciphering algorithm. The formal security definition requires that the domain of the cipher to be split into two sets, so $\mathcal{D} = \mathcal{D}_L \times \mathcal{D}_R$. We may refer to $\mathcal{D}_L$ as the *left set,* and $\mathcal{D}_R$ as the *right set*; henceforth, we will let $\mathcal{D}_L = \{0,1\}^n$ and $\mathcal{D}_R = \{0,1\}^m$ for some positive integers n and m such that $n < m$. We denote a tweakable split-domain cipher by its enciphering algorithm

$$\mathsf{EE} : \{0,1\}^k \times \mathcal{H} \times \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n \times \{0,1\}^m ,$$

where, for any $K \in \{0,1\}^k$ and any $H \in \mathcal{H}$, the function EE_K^H identifies a permutation over $\{0,1\}^n \times \{0,1\}^m$, whose inverse we denote by ED_K^H. Thus, classical tweakable blockciphers can be viewed as the special case where $m = 0$ or, equivalently, $\mathcal{D}_R = \emptyset$.

In order to make CGO forward secure, we augment tweakable split-domain ciphers to also be *updatable* so that their key material can be refreshed. Formally, an updatable tweakable split-domain cipher consists of a pair of functions $(\mathsf{EE}, \mathsf{EU})$ where EE is a tweakable split-domain cipher and the update algorithm EU takes a key $K \in \{0,1\}^k$ and a left element $X_L \in \{0,1\}^n$, and returns new values for each, i.e.,

$$\mathsf{EU} : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^k \times \{0,1\}^n \ .$$

In terms of security, we will require that an updatable tweakable split-domain cipher satisfies an extended security notion called URRND, where the U stands for "updatable" and RRND is the two-sided random function equivalent of RPRP. Degabriele and Karadžić already introduced RRND, which turns out a lot more convenient to work with. The formal definition of URRND security is deferred to Section 4, where we also describe a construction for an updatable tweakable split-domain cipher that meets this notion (Section 4.2) and that can thus be plugged in directly to instantiate CGO.

3 The Design of CGO

3.1 Context

Tor. The Tor network is an overlay network that allows users, running an onion proxy (OP), to establish a circuit involving multiple onion routers (ORs), typically three. A Tor circuit allows traffic to flow in both directions: forward from the proxy to any of the routers, or backward from any of the routers to the proxy. Traffic is split up in fixed size messages that are individually sent across the circuit using cells. Each Tor cell consists of a cell header and a cell payload, where only the latter part is encrypted. Thus onion routing involves cells, whereas the onion encryption we are interested in only deals with the cell's payload (which we will refer to as the ciphertext). For all Tor versions so far, the length of these ciphertexts is fixed at 509 bytes (Tor's CELL_BODY_LEN).

The cell header indicates the type of cell, of which we are primarily concerned with the types RELAY and RELAY_EARLY. In turn, the payload of these cells is itself partitioned into 11 bytes of header fields and a data field of 498 = 509 - 11 bytes, where the actual message is contained, possibly augmented with padding. Currently, the headers in the payload of a RELAY cell consist of the following: a 1-byte Relay Command field, a 2-byte Recognised field, a 2-byte Stream Identifier field, a 4-byte Digest field, and a 2-byte Length field. Encryption uses counter mode, with the Recognised and Digest fields simultaneously providing end-to-end integrity and enabling leaky pipes (the ability for the proxy to communicate with any of the routers).

Our new scheme, CGO, was explicitly designed to be used in Tor, and in particular, it aims to fit Tor's existing infrastructure and facilitate its deployment with only some mild alterations. Indeed, CGO preserves all of Tor's currently existing functionality while augmenting it with new features. However, the exact mechanisms by which it realises Tor's existing functionality, such as leaky pipes and end-to-end integrity, is different, which requires changes to the cell payload format. As we will see momentarily, CGO views a ciphertext as consisting of two parts, $C = T \parallel \overline{C}$, where the functionality previously provided by the Recognised and Digest fields is now provided by T—acting as a 16-byte authentication tag. Accordingly, in CGO we dispose of these two fields from the payload and relocate the remaining components of the payload in $\overline{C}$. This reduces the typical supported payload length of a relay cell from 498 bytes to 488 = 509 - 5 - 16 bytes while also raising the cell integrity from 32 bits, previously provided by the Digest field, to 128 bits of security that is now provided by T.

Syntax. Although initially the forward and backward directions might appear symmetrical, the key generation is initiated and coordinated by the proxy, creating an asymmetry. Furthermore, the introduction of leaky pipes substantially increases the degree of asymmetry between the two directions.

Specifically, in the forward direction a router needs to determine whether an incoming ciphertext is intended for itself or needs to be forwarded further. Locally stored routing information is insufficient to make this determination, instead cryptographic processing of the ciphertext at an onion router will be needed to decide. Conversely, in the backward direction, in addition to processing cells passing through the circuit, any onion router can send messages to the proxy. Consequently, the onion proxy will need to determine from which onion router a received message originated.

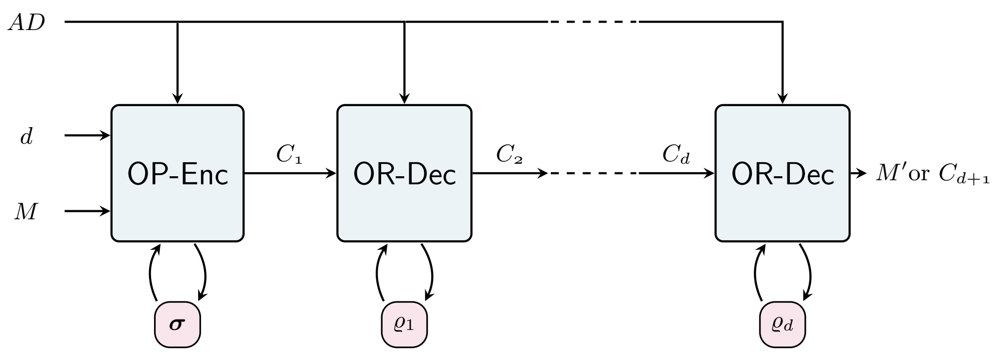

Fig. 1. The syntax of onion encryption in the forward direction.

When specifying CGO below, we will slightly simplify our presentation of the circuit initialization. Whereas Tor uses a telescoping mechanism to use the existing part of the circuit to extend it one router at a time, we will pretend that all parties' states are initialized simultaneously without further communication necessary. This simplification is common when modelling onion encryption or routing [15, 41] and our specification of CGO is of course fully compatible with Tor's telescoping.

Notation. For a circuit of length ℓ, we assign the router nodes indices $1, \ldots, \ell$ starting from the node adjacent to the proxy up to the last node in the circuit. For any node i within a circuit we refer to node $i-1$ as its predecessor and node $i+1$ as its successor. Here the proxy can be thought of as a node with index 0. We assume that the message M to be communicated is in the message space $\mathcal{M} = \{0,1\}^{\mathsf{m}}$, i.e., messages are padded to a fixed length using some injective padding, and that the ciphertexts communicated between routers and the proxy are in the ciphertext space $\mathcal{C} = \{0,1\}^{\mathsf{c}}$. Furthermore, we assume $\mathsf{m} < \mathsf{c}$ so that $\mathcal{M}$ and $\mathcal{C}$ are disjoint and thus messages can be distinguished from ciphertexts through their length. In the case of Tor, we will use $\mathsf{c} = 8 \cdot 509$ and $\mathsf{m} = 8 \cdot (509 - 16)$ (treating the 5 bytes of payload header as part of the message space given they need to be encrypted). Associated data is modelled by $\mathcal{AD} \subseteq \{0,1\}^*$. For Tor, we will assume associated data has a fixed length, typically $\mathcal{AD} = \{0,1\}^8$ (see Section 3.3). Ciphertexts between node $i-1$ and i may be indexed by i, so in the forward direction router i will take as input C_i whereas in the backward direction router i will output C_i.

Forward direction. A forward-direction onion encryption scheme OE-Fw consists of three algorithms (Init-Fw, OP-Enc, OR-Dec). A pictorial representation of their operation and lifecyle of a message in transit is shown in Fig. 1.

– The circuit initialisation algorithm Init-Fw takes as input the circuit size ℓ and returns the initial encryption state $\boldsymbol{\sigma}$ intended for the proxy (which consists of ℓ components σ_i, one for each router on the circuit), as well as the initial decryption states $\varrho_1, \ldots, \varrho_\ell$ for each router in the circuit.
– The deterministic algorithm OP-Enc is used by the proxy to encrypt messages with associated data to a specific node within a circuit. Given the current encryption state $\boldsymbol{\sigma}$ for a circuit, the index $d \in [\ell]$ of the destination node within that circuit, associated data $AD \in \mathcal{AD}$, and a message $M \in \mathcal{M}$, the call $C_1 \leftarrow \mathsf{OP\text{-}Enc}^{AD}_{\langle\boldsymbol{\sigma}\rangle}(d, M_j)$ returns the ciphertext $C_1 \in \mathcal{C}$ intended for the first available router; as a side-effect the algorithm OP-Enc may update its state $\boldsymbol{\sigma}$.
– The deterministic algorithm OR-Dec is used by the routers in the circuit to process incoming ciphertexts. It takes as input the node's decryption state ϱ_i, the associated data AD, and a ciphertext $C_i \in \mathcal{C}$ to update its state and output $C_{i+1} \in \mathcal{M} \cup \mathcal{C}$. If $C_{i+1} \in \mathcal{M}$ the output is deemed intended for the local node, otherwise it is supposed to be forwarded to the next node in the circuit.
(In the more general setting, where possibly $\mathcal{M} \cap \mathcal{C} \neq \emptyset$, OR-Dec's syntax should be adjusted to output a bit indicating whether the ciphertext has been recognised as destined for the current router or is supposed to be forwarded.)

We do not explicitly model any ciphertext rejection, as we expect a forged ciphertext to propagate through the circuit without being recognised by any of its nodes. The last router in the circuit cannot forward any output $C_{\ell+1} \in \mathcal{C}$ and thus, at its routing level will have to take an appropriate action, such as initiating a teardown of the circuit. This separation of routing cells versus the cryptographic processing of their ciphertexts allows for the latter process to be performed by a router agnostic of whether it is the last node in the circuit.

Backwards direction. A backward-direction onion encryption scheme OE-Bw consists of four algorithms (Init-Bw, OR-Enc, OR-Proc, OP-Dec).

- The circuit initialisation algorithm Init-Bw takes as input the circuit size ℓ and returns the initial decryption state $\boldsymbol{\sigma}$ intended for the proxy (which consists of ℓ components σ_i, one for each router on the circuit), as well as the initial encryption/processing states $\varrho_1, \ldots, \varrho_\ell$ for each router in the circuit.
- On the one hand, the deterministic algorithm OR-Enc is used by a router to encrypt a fresh message, intended for the proxy. It takes the router's encryption state ϱ_i, associated data AD, and a message M to update its state and output a ciphertext C_i, to be forwarded to its predecessor node in the circuit.
- On the other hand, the deterministic algorithm OR-Proc is used by a router to process a ciphertext (on its way to the proxy) that was forwarded by its successor node in the circuit. Given the node's encryption state ϱ_i for the circuit, associated data AD, and an input ciphertext C_{i+1} the algorithm OR-Proc updates its encryption state and returns a new ciphertext C_i (intended for the predecessor node in the circuit).
- Finally, the algorithm OP-Dec is used by the proxy to decrypt an incoming ciphertext and determine its origin. It takes as input the decryption state $\boldsymbol{\sigma}$, associated data $AD \in \mathcal{AD}$, and a ciphertext C to return a string X, and a symbol $s \in \{0, 1, \ldots, \ell\}$. If $s > 0$ then X must be a message in $\mathcal{M}$ and s indicates the node in the circuit from which the ciphertext originated. Alternatively, if $s = 0$ the ciphertext has been deemed invalid and $X = \perp$. Practically, $\perp$ may be represented by any convenient string, e.g. the empty string, and more generally, X could also be a string encoding protocol information such as a specific error message.

3.2 Specification

We provide high-level pseudocode descriptions of our new scheme Counter Galois Onion (CGO) in the forward and backward directions in Figs. 2 and 3, respectively. This high-level description is stated in terms of an updatable tweakable split-domain cipher EE with update function EU. The message space $\mathcal{M} = \{0,1\}^m$ and each ciphertext consists of two strings T and C of sizes n and m respectively (thus $\mathcal{C} = \{0,1\}^{n+m}$; for completeness, we furthermore require $\mathcal{H} = \{0,1\}^n \times \mathcal{AD}$).

Roughly speaking, each layer of encryption in CGO then corresponds to an enciphering or deciphering operation of the split-domain cipher. Perhaps counterintuitively, in the forward direction we use deciphering at the OP to encrypt a message (and create an initial ciphertext), which is followed by enciphering at the ORs to decrypt (peeling off the layers). In contrast, in the backward direction we use enciphering at the ORs to encrypt a message (or process a ciphertext) and deciphering to decrypt at the OP.

To each encryption layer, CGO associates a state consisting of the updatable tweakable split-domain cipher's key K, a nonce N, and a string T' initialised to the all-zero string. Thus for both directions, every OR in the circuit will maintain such a triple as its state. Likewise the state of the OP will consist of the aggregate of triples for each layer (OR in the circuit) and both directions.

Shortly, we will explain the rationale behind CGO's design, keeping coverage of its security properties informal. In Section 4.2 we describe how to create the underlying abstract building block from a tweakable cipher and a pseudorandom function, and then finish the full specification of CGO by suggesting concrete instantiations based on AES.

3.3 Design Rationale

Non-malleability only where it is needed. A central goal of CGO is to protect against tagging attacks that exploit the malleability in Tor's encryption layers [19]. Naturally, protecting against these attacks requires that the underlying primitive behind each encryption layer be non-malleable. Authenticated encryption, is non-malleable (by virtue of its IND-CCA security), yet necessarily introduces redundancy in its ciphertexts. Unfortunately, ciphertext expansion is detrimental to onion encryption as an adversary can infer the relative position of a node in a circuit from the size of the ciphertexts flowing in and out of that node [15, 33]. A better alternative is a tweakable wide-block cipher that is secure as a strong pseudorandom permutation, as it provides non-malleability without incurring any ciphertext expansion. Indeed, proposal 261 [34] adopts this approach, which does suffice to prevent tagging attacks [15].

CGO refines this approach by replacing the tweakable wide-block cipher with an updatable tweakable split-domain cipher that only needs to be URRND secure, a strictly weaker security requirement. As a result, CGO admits tweakable cipher constructions with better performance characteristics.

```
$\mathsf{Init\text{-}Fw}(\ell)$
for $i = 1$ to $\ell$
    $K \leftarrow\!\!\$\ \{0,1\}^k$
    $N \leftarrow\!\!\$\ \{0,1\}^n$
    $T' \leftarrow 0^n$
    $\sigma_i \leftarrow (K, N, T')$
    $\varrho_i \leftarrow (K, N, T')$
$\boldsymbol{\sigma} \leftarrow (\sigma_1, \ldots, \sigma_\ell)$
return $(\boldsymbol{\sigma}, \varrho_1, \ldots, \varrho_\ell)$
```

```
$\mathsf{OP\text{-}Enc}^{AD}_{\langle\boldsymbol{\sigma}\rangle}(d, M)$
if $(d > \ell)$ return $\perp$
for $i = d$ to $1$
    $(K, N, T') \twoheadleftarrow \sigma_i$
    $H \leftarrow T' \parallel AD$
    if $i = d$    // initialize top layer
        $(T, \overline{C}) \leftarrow (N, M)$
    $T' \leftarrow T$
    $(T, \overline{C}) \leftarrow \mathsf{ED}^H_K(T, \overline{C})$
    if $i = d$
        // update destination's key
        $(K, N) \leftarrow \mathsf{EU}_K(N)$
    $\sigma_i \leftarrow (K, N, T')$
return $T \parallel \overline{C}$
```

```
$\mathsf{OR\text{-}Dec}^{AD}_{\langle\varrho\rangle}(T \parallel \overline{C})$
$(K, N, T') \twoheadleftarrow \varrho$
$H \leftarrow T' \parallel AD$
$(T, \overline{C}) \leftarrow \mathsf{EE}^H_K(T, \overline{C})$
if $T = N$
    // ciphertext recognised
    $(K, N) \leftarrow \mathsf{EU}_K(N)$
    $X \leftarrow \overline{C}$
else
    // forwarding ciphertext
    $X \leftarrow T \parallel \overline{C}$
$\varrho \leftarrow (K, N, T)$
return $X$
```

Fig. 2. CGO in the forward direction.

```
$\mathsf{Init\text{-}Bw}(\ell)$
for $i = 1$ to $\ell$
    $K \leftarrow\!\!\$\ \{0,1\}^k$
    $N \leftarrow\!\!\$\ \{0,1\}^n$
    $T' \leftarrow 0^n$
    $\sigma_i \leftarrow (K, N, T')$
    $\varrho_i \leftarrow (K, N, T')$
$\boldsymbol{\sigma} \leftarrow (\sigma_1, \ldots, \sigma_\ell)$
return $(\boldsymbol{\sigma}, \varrho_1, \ldots, \varrho_\ell)$
```

```
$\mathsf{OP\text{-}Dec}^{AD}_{\langle\boldsymbol{\sigma}\rangle}(T \parallel \overline{C})$
$i \leftarrow 1,\ s \leftarrow 0$
while $i \leq \ell\ \wedge\ s = 0$
    $(K, N, T') \twoheadleftarrow \sigma_i$
    $H \leftarrow T' \parallel AD$
    $(T', \overline{C}) \leftarrow \mathsf{ED}^H_K(T, \overline{C})$
    if $T' = N$
        // ciphertext recognised
        $(K, N) \leftarrow \mathsf{EU}_K(T)$
        $s \leftarrow i; M \leftarrow \overline{C}$
    $\sigma_i \leftarrow (K, N, T)$
    $i \leftarrow i + 1,\ T \leftarrow T'$
if $s = 0$
    $M \leftarrow \perp$
return $(M, s)$
```

```
$\mathsf{OR\text{-}Enc}^{AD}_{\langle\varrho\rangle}(M)$
$(K, N, T') \twoheadleftarrow \varrho$
$H \leftarrow T' \parallel AD$
$(T, \overline{C}) \leftarrow \mathsf{EE}^H_K(N, M)$
$(K, N) \leftarrow \mathsf{EU}_K(T)$
$\varrho \leftarrow (K, N, T)$
return $T \parallel \overline{C}$
```

```
$\mathsf{OR\text{-}Proc}^{AD}_{\langle\varrho\rangle}(T \parallel \overline{C})$
$(K, N, T') \twoheadleftarrow \varrho$
$H \leftarrow T' \parallel AD$
$(T, \overline{C}) \leftarrow \mathsf{EE}^H_K(T \parallel \overline{C})$
$\varrho \leftarrow (K, N, T)$
return $T \parallel \overline{C}$
```

Fig. 3. CGO in the backward direction.

Informally, the main observation supporting that URRND security suffices to thwart tagging attacks is as follows: only the processing at the ORs needs to be non-malleable, whereas that at the OP does not. Inherited from Degabriele and Karadžić's RPRP security notion, URRND security involves asymmetric properties of the enciphering and deciphering algorithms of the tweakable split-domain cipher. Loosely speaking, the enciphering algorithm is fully non-malleable, whereas the deciphering algorithm only offers a very restricted form of non-malleability (see Definition 1 in Section 4.1 for details). Accordingly, CGO always uses the enciphering algorithm at the ORs, irrespective of the direction of the traffic flow, as there the processing of ciphertexts needs to be non-malleable. The weaker deciphering algorithm offers sufficient security for processing at the OP, both for encryption and decryption.

The role of the nonce. CGO encrypts messages together with a nonce N that is selected at random. Normally, the nonce serves to diversify ciphertexts (ensuring that repeating messages results in distinct and random-looking ciphertexts) and, when transforming a conventional tweakable cipher into an AEAD scheme, the nonce can be included in the tweak without causing any expansion. However, for CGO we employ a variant of the encode-then-encipher paradigm [8, 13, 43] where the nonce is included as part of the 'plaintext' input of the cipher and simultaneously provides ciphertext diversification and integrity.

As URRND security is only effective as long as the left input to the deciphering algorithm does not repeat, that left input should include the nonce. At first sight, CGO's handling of the nonce may seem sub-optimal due to the resulting ciphertext expansion. CGO compensates for this expansion by using the nonce's redundancy to provide end-to-end plaintext integrity (in the operations comparing T or T' to N during decryption), thereby resulting in no additional net overhead. Intuitively, this dual use of the nonce is possible because URRND security ensures that the left output of both the deciphering and enciphering algorithms is unpredictable for unused inputs.

Supporting leaky pipes. In Tor, the OP can send messages to *any* OR in the circuit, and conversely, any OR can send messages to the OP. This functionality, informally called *leaky pipes* within the Tor ecosystem, poses a further design challenge that CGO needs to support. Indeed, another reason for using the nonce to achieve end-to-end integrity is that it accommodates leaky pipes. Namely, in CGO, a distinct nonce is associated with each encryption layer, which is shared and maintained synchronously by the OP and the corresponding OR. In the forward direction, during encryption, the OP will embed the nonce corresponding to the OR for which the message is intended, apply the appropriate layers of encryption, and update the nonce. Then, as the resulting ciphertext travels through the circuit, each OR will decrypt the ciphertext and check whether the left part of the output matches the locally stored copy of the nonce. If this check succeeds, the ciphertext is understood to be intended for that OR, the right output is returned as the authenticated message, and the nonce is updated. Alternatively, if the check fails, the complete decryption output (both the left and right parts) is returned as the output ciphertext and forwarded to the next OR in the circuit. In the backward direction, an analogous process occurs whereby the source OR encrypts the message together with the nonce, and each subsequent OR adds another layer of encryption. Then the OP decrypts the ciphertext iteratively, layer by layer, each time comparing the left part of the decrypted output to the nonce for that layer. Decryption halts either when there is a match whereby the message and the source OR are recovered and the corresponding nonce is subsequently updated, or all layers are exhausted, in which case decryption has failed and none of the nonces is updated.

Ciphertext chaining. By itself, requiring the encryption layers to be non-malleable does not exclude all types of tagging attacks [15]. An adversary can always carry out a rudimentary form of tagging attack, where the adversary tests whether two ORs, which it has access to, lie on the same circuit or not. Namely, the adversary can tamper a cell as the first OR sends it out and then observe whether decryption fails at the receiver OR. Any scheme that provides integrity only across its endpoints succumbs to this attack making it impossible to prevent without incurring further ciphertext expansion. However, a good scheme ensures that this attack cannot be repeated across different cells on the same circuit, as that would effectively enable the adversary to carry out the equivalent of a full-fledged tagging attack by encoding the tag over a sequence of cells. CGO protects against this by ensuring that any tampered cell entering an OR drives its state out-of-sync from the OP in an irreversible manner. As a result, all subsequent cells output by that OR will be randomised, and as they propagate further down the circuit, they will irreversibly drive subsequent ORs out-of-sync as well. Thus, as long as there is a single OR that is not under adversarial control, the circuit cannot recover from a decryption failure even if the adversary can rewind the recipient's state.

CGO achieves this domino effect by chaining the processing of consecutive ciphertexts at every node in the circuit. Specifically, at every OR, the left part of the prior output (the tag) is included in the tweak when processing the subsequent input pair of strings. A matching stateful operation is performed at the OP. This mechanism is similar in spirit to that used in proposal 261, which included the XOR of the (full) prior input and output in the tweak. However, CGO's approach is more efficient as it only needs to process an extra 16 bytes of tag in the tweak instead of 512 bytes. To achieve the desired effect, it is crucial to chain the tag *output* by the OR, which depends on all input bits, including the associated data.

Another novel feature of CGO is its support for associated data. In Tor, the cell header, which is the only part of the cell that is unencrypted, consists of a one-byte command field and a two-byte circuit identifier. However the latter is a mutable field and thus only the command field can be included in the associated data. Nevertheless this still serves to mitigate against tagging attacks that tamper with cell headers, such as changing the command field from RELAY to RELAY_EARLY, which has been exploited in the past [27,42].

Forward security. The chaining of ciphertexts is not the only stateful mechanism in CGO. To achieve forward security, CGO includes a rekeying mechanism to refresh the key material of the end nodes every time a message is transmitted. In contrast, the key material of the intermediate nodes is not updated. Instead of introducing a separate primitive to provide this rekeying, CGO assumes that the tweakable split-domain cipher includes this functionality, which is precisely what the *update* functionality described in Section 2.1 is intended for. Additionally, CGO uses the update functionality to generate unpredictable nonces in a stateful way. The full CGO specification instantiates the updatable split-domain tweakable cipher with a variant of the UIV construction [13], see Section 4.2.

Supporting heterogeneity. As Tor is a distributed protocol, upgrading its relay cryptography poses additional challenges. In particular, some degree of compatibility is required between the new cryptographic protocol and the existing one to facilitate gradual migration, as upgrading all of the nodes in the Tor network simulaneously is infeasible. Thus, for a transition to be tenable, operating a heterogeneous network, where only a fraction of the nodes run CGO, is inevitable. Fortunately, even though the supported payload sizes do not match, operating such a heterogeneous network, where only a subset of the ORs run CGO is possible, provided the OP knows which ORs in the circuit support CGO (this information could be obtained during circuit establishment or even before, when gathering data about the available Onion Routers). Any heterogeneous circuit with at least one (honest) router running CGO will already offer increased protection against tagging attacks. Of course, if the OP itself does not support CGO, then the circuit must necessarily be (old-school) homogeneous.

4 Updatable Tweakable Split-Domain Ciphers

For added generality and ease of exposition, we presented CGO in a top-down, modular fashion, describing CGO's working in terms of an updatable tweakable split-domain cipher as the main underlying primitive. Our security analysis of CGO follows a similar approach, thereby allowing us to instantiate the cipher with a different one while retaining the same security guarantees. In the first part of this section we develop and link together a sequence of security notions of increasing complexity, for an updatable tweakable split-domain cipher. We then go on to present the UIV+ construction and its concrete instantiation GCM-UIV+ we suggest as the updatable tweakable split-domain cipher in the concrete specification of CGO.

4.1 Security Notions

Degabriele and Karadžić introduced the notion of a rugged pseudorandom permutation (RPRP) [13] as a weakening of the strong pseudorandom permutation notion for wide-block ciphers. The security definition itself requires that the domain of the wide-block cipher be split into two parts, and their use of the term RPRP refers both to the security notion itself as well as any split-domain cipher meeting this notion. Informally, RPRP security requires that access to the tweakable split-domain cipher be indistinguishable from access to a family of ideal permutations Π over the same split domain, where the

```
Experiment $\mathsf{Exp}^{\mathrm{URRND}\text{-}b}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A})$
  $K \leftarrow\!\$\ \{0,1\}^k$
  $\hat{b} \leftarrow \mathbb{A}^{\mathrm{En},\mathrm{De},\mathrm{Gu},\mathrm{Up}}$
  return $\hat{b}$

En$(H, X_L, X_R)$
  if $b = 1$
    $(Y_L, Y_R) \leftarrow \mathsf{EE}^H_K(X_L, X_R)$
  else
    $(Y_L, Y_R) \leftarrow\!\$\ \mathcal{D}_L \times \mathcal{D}_R$
  $\mathcal{F} \xleftarrow{\cup} Y_L,\ \mathcal{U} \xleftarrow{\cup} (H, Y_L, Y_R)$
  return $(Y_L, Y_R)$

Gu$(H, Y_L, Y_R, X'_L)$
  if $(H, Y_L, Y_R) \in \mathcal{U}$
    return $\lightning$
  if $b = 1$
    $(X_L, X_R) \leftarrow \mathsf{ED}^H_K(Y_L, Y_R)$
    return $X_L = X'_L$
  else
    return false

Experiment $\mathsf{Exp}^{\mathrm{RRND}\text{-}b}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A})$
  $K \leftarrow\!\$\ \{0,1\}^k$
  $\hat{b} \leftarrow \mathbb{A}^{\mathrm{En},\mathrm{De},\mathrm{Gu}}$
  return $\hat{b}$

De$(H, Y_L, Y_R)$
  if $Y_L \in \mathcal{F}$
    return $\lightning$
  if $b = 1$
    $(X_L, X_R) \leftarrow \mathsf{ED}^H_K(Y_L, Y_R)$
  else
    $(X_L, X_R) \leftarrow\!\$\ \mathcal{D}_L \times \mathcal{D}_R$
  $\mathcal{F} \xleftarrow{\cup} Y_L,\ \mathcal{U} \xleftarrow{\cup} (H, Y_L, Y_R)$
  return $(X_L, X_R)$

Up$(T)$
  if $b = 1$
    $(\overline{K}, \overline{T}) \leftarrow \mathsf{EU}_K(T)$
  else
    $(\overline{K}, \overline{T}) \leftarrow\!\$\ \{0,1\}^k \times \mathcal{D}_L$
  return $(\overline{K}, \overline{T})$
```

Fig. 4. The $\mathsf{Exp}^{\mathrm{URRND}\text{-}b}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A})$ and $\mathsf{Exp}^{\mathrm{RRND}\text{-}b}_{\mathsf{EE}}(\mathbb{A})$ experiments used to define URRND and RRND security for an (updatable) tweakable split-domain cipher.

tweak is used to access different permutations. However, while the adversary has unfettered access to the encipher functionality, its interaction with the decipher functionality is severely restricted: the adversary can access the decipher functionality via two separate oracles which offer distinct interfaces and impose different restrictions (details follow).

Degabriele and Karadžić considered a second security notion, called RRND security, where the family of ideal permutations in RPRP is replaced by a tweakable two-sided random function, and showed that it implies RPRP security by leveraging an earlier result of Halevi and Rogaway [26, Lemma 6]. More concretely, RRND and RPRP security are equivalent up to a birthday bound and it turns out that RRND is actually the easier notion to work with, both to prove that a smaller construction is RRND secure, and that a higher level construction building on a tweakable split-domain cipher inherits its security (using RPRP security instead would simply incur the birthday-style loss twice, moving to and fro).

Our first step is to extend the RRND security notion for the case of an *updatable* tweakable split-domain cipher, resulting in the extended notion URRND. We show that any tweakable split-domain cipher satisfying RPRP security can be turned into an updatable tweakable split-domain cipher satisfying URRND security by augmenting it with a pseudorandom update functionality. In the URRND game, however, the tweakable split-domain cipher does not get queried on the updated keys, as is the case in CGO. Accordingly, our second and more elaborate variant of RRND security, called CRND$_\ell$, captures the security of an updatable tweakable split-domain cipher under *continuous* key updates, where ℓ indicates the number of initial keys that can all be updated independently. We next present the formal definitions of these two security notions.

The URRND notion. The URRND experiment for an updatable tweakable split-domain cipher $(\mathsf{EE}, \mathsf{EU})$ is described in Fig. 4, where the adversary is given access to an encipher oracle En, a decipher oracle De, a guess oracle Gu, and an update oracle Up. The encipher oracle is queried on an input pair (X_L, X_R) and a tweak H, and depending on the value of the bit b, it either evaluates this input using the updatable tweakable split-domain cipher EE or a lazily sampled two-sided random function to return an output pair (Y_L, Y_R). The decipher oracle De works analogously, but it can only be queried on inputs (H, Y_L, Y_R) with a *fresh* left-component Y_L. If Y_L has either been returned by En or been input to De before, the new De-query will simply refuse. Notably, freshness is *not* bound to the tweaks, thus the query $\mathrm{De}(H, Y_L, Y_R)$ cannot be followed by $\mathrm{De}(H', Y_L, Y'_R)$ for any values of H' and Y'_R. The freshness requirement on Y_L is enforced via the set $\mathcal{F}$. Additionally, the adversary can interact with the decipher functionality via the oracle Gu to guess whether deciphering a chosen input would result in a specific guessed left value. However, when $b = 0$ this oracle will always return false. Accordingly, queries to the guess queries must be *unused*, i.e., the query must not contain a triple (H, Y_L, Y_R) that was either returned by En or previously queried to De. This requirement is enforced through the set $\mathcal{U}$. Finally the adversay has access to an update oracle that either returns the output of the real update algorithm Up, when $b = 1$, or a randomly sampled output if $b = 0$.

Without loss of generality, we consider adversaries that do not make *pointless queries*, thus they never repeat a query to any of their oracles and, if the query $(Y_L, Y_R) \leftarrow \mathrm{En}(H, X_L, X_R)$ was made, the query $\mathrm{De}(H, Y_L, Y_R)$ is never made, and vice versa. (If pointless queries were to be allowed, extra game administration would be needed to ensure consistent answers in the ideal setting.)

The RRND experiment with respect to a tweakable split-domain cipher EE is defined in the exact same way, except that the adversary does not have access to the update oracle. Compared to the original RRND definition [13], we assume that the tweakable split-domain cipher has a fixed-size domain, and we have restricted guess queries to include only a single guess.

Definition 1 (URRND **and** RRND **Advantages).** *For any tweakable split-domain cipher* EE *over* $\mathcal{D}_L \times \mathcal{D}_R$ *and any update function* EU*, the corresponding* URRND *and* RRND *advantages of an adversary* $\mathbb{A}$ *are given by:*

$$\mathsf{Adv}^{\mathrm{URRND}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = \Pr\left[\mathsf{Exp}^{\mathrm{URRND}\text{-}0}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathrm{URRND}\text{-}1}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\right]$$
$$\mathsf{Adv}^{\mathrm{RRND}}_{\mathsf{EE}}(\mathbb{A}) = \Pr\left[\mathsf{Exp}^{\mathrm{RRND}\text{-}0}_{\mathsf{EE}}(\mathbb{A}) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathrm{RRND}\text{-}1}_{\mathsf{EE}}(\mathbb{A}) = 1\right] ,$$

where $\mathsf{Exp}^{\mathrm{URRND}\text{-}b}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A})$ *and* $\mathsf{Exp}^{\mathrm{RRND}\text{-}b}_{\mathsf{EE}}(\mathbb{A})$ *are defined in Fig. 4.*

The continuous-key-updates CRND$_\ell$ notion. We now consider a variant of the URRND game, denoted CRND$_\ell$, where we extend the URRND game to a multi-instance setting. Firstly, the adversary can interact with ℓ independent instances of the updatable split-domain tweakable cipher. Secondly, for every cipher instance, its key K_h can be updated via the Up oracle, and only the updated T is returned to the adversary. For each cipher instance h, the index i_h keeps track of the latest version of its key. Thus, every query to the oracles En, De, Gu will now include both an instance handle h and an index j indicating which version of the key for that instance the oracle should use. Finally, we allow the adversary to corrupt cipher instances. Namely, the adversary can use the oracle Co to obtain the latest key version $K_{h[i_h]}$ of any cipher instance h. However, an instance can only be corrupted if its latest key is fresh, meaning it has not been used in a prior query to the oracles En, De, Gu, and once corrupted, its key can no longer be updated.

Definition 2 (CRND$_\ell$ **Advantage).** *For any tweakable split-domain cipher* EE, EU *over* $\mathcal{D}_L \times \mathcal{D}_R$ *and any update function* EU*, the corresponding* CRND$_\ell$ *advantage of an adversary* $\mathbb{A}$ *is given by:*

$$\mathsf{Adv}^{\mathrm{CRND}_\ell}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = \Pr\left[\mathsf{Exp}^{\mathrm{CRND}_\ell\text{-}0}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathrm{CRND}_\ell\text{-}1}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\right] ,$$

where $\mathsf{Exp}^{\mathrm{CRND}_\ell\text{-}b}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A})$ *is defined in Fig. 5.*

We analyse the relation between the RRND, URRND, and CRND$_\ell$ security notions in Appendix A.

4.2 The UIV+ Construction Used in CGO

In CGO, the updatable tweakable split-domain cipher is instantiated by UIV+ (see Fig. 6). Its encipher and decipher algorithms are inherited from UIV [13], whereas the update functionality is novel, reusing UIV's components. UIV+ is composed of a tweakable blockcipher E with inverse D, and a tweakable pseudorandom function F. The header H and the right input X_R are injectively mapped to form E's tweak. If the size of H is fixed, as is the case in Tor, concatenation of H and X_R yields an injective mapping to the tweak. Besides its key, the tweakable pseudorandom function F takes two inputs, a tag

Experiment $\mathsf{Exp}^{\mathrm{CRND}_\ell\text{-}b}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A})$

```
for $h = 1$ to $\ell$
    $K_{h[0]} \leftarrow_{\$} \{0,1\}^k$
    $fresh_h \leftarrow \mathsf{true}$
    $i_h \leftarrow 0$
$\hat{b} \leftarrow \mathbb{A}^{\mathrm{En,De,Gu,Up,Co}}$
return $\hat{b}$
```

$\mathrm{En}(h, j, H, X_L, X_R)$

```
if $(j > i_h) \vee (j = i_h \wedge h \in \mathcal{Z})$
    return $\lightning$
if $j = i_h$
    $fresh_h \leftarrow \mathsf{false}$
if $b = 1$
    $(Y_L, Y_R) \leftarrow \mathsf{EE}^H_{K_{h[j]}}(X_L, X_R)$
else
    $(Y_L, Y_R) \leftarrow_{\$} \mathcal{D}_L \times \mathcal{D}_R$
$\mathcal{F}^j_h \xleftarrow{\cup} Y_L,\ \mathcal{U}^j_h \xleftarrow{\cup} (H, Y_L, Y_R)$
return $(Y_L, Y_R)$
```

$\mathrm{Gu}(h, j, H, Y_L, Y_R, X'_L)$

```
if $(j > i_h) \vee (j = i_h \wedge h \in \mathcal{Z}) \vee (H, Y_L, Y_R) \in \mathcal{U}^j_h$
    return $\lightning$
if $j = i_h$
    $fresh_h \leftarrow \mathsf{false}$
if $b = 1$
    $(X_L, X_R) \leftarrow \mathsf{ED}^H_{K_{h[j]}}(Y_L, Y_R)$
    return $X_L = X'_L$
else
    return $\mathsf{false}$
```

$\mathrm{De}(h, j, H, Y_L, Y_R)$

```
if $(j > i_h) \vee (j = i_h \wedge h \in \mathcal{Z}) \vee (Y_L \in \mathcal{F}^j_h)$
    return $\lightning$
if $j = i_h$
    $fresh_h \leftarrow \mathsf{false}$
if $b = 1$
    $(X_L, X_R) \leftarrow \mathsf{ED}^H_{K_{h[j]}}(Y_L, Y_R)$
else
    $(X_L, X_R) \leftarrow_{\$} \mathcal{D}_L \times \mathcal{D}_R$
$\mathcal{F}^j_h \xleftarrow{\cup} Y_L,\ \mathcal{U}^j_h \xleftarrow{\cup} (H, Y_L, Y_R)$
return $(X_L, X_R)$
```

$\mathrm{Up}(h, T)$

```
if $h \in \mathcal{Z}$
    return $\lightning$
if $b = 1$
    $(K_{h[i_h+1]}, T) \leftarrow \mathsf{EU}_{K_{h[i_h]}}(T)$
else
    $K_{h[i_h+1]} \times T \leftarrow_{\$} \{0,1\}^k \times \mathcal{D}_L$
$i_h \leftarrow i_h + 1, fresh_h \leftarrow \mathsf{true}$
return $T$
```

$\mathrm{Co}(h)$

```
if $(h \in \mathcal{Z}) \vee (fresh_h = \mathsf{false})$
    return $\lightning$
$\mathcal{Z} \xleftarrow{\cup} h$
return $K_{h[i_h]}$
```

Fig. 5. The $\mathsf{Exp}^{\mathrm{CRND}_\ell\text{-}b}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A})$ experiment used to define CRND_ℓ security instances for ℓ independent instances of an updatable tweakable split-domain cipher with continuous key updates.

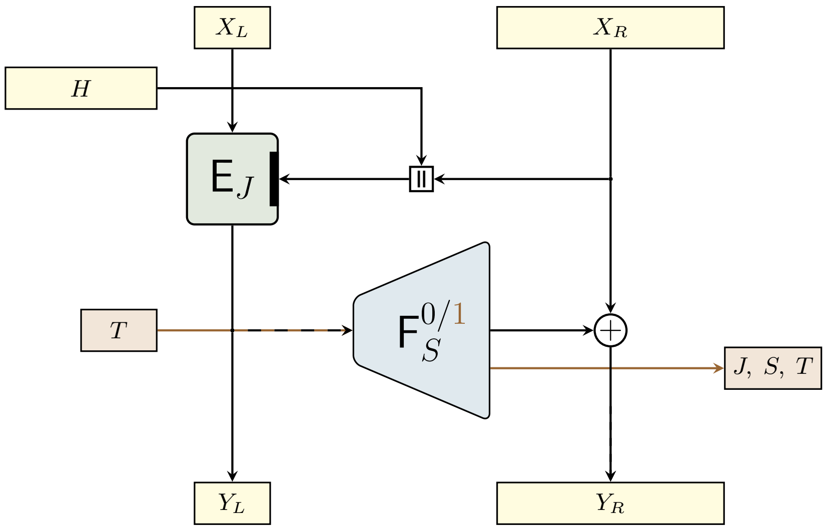

Fig. 6. The UIV+ construction used to instantiate the URRND-secure updatable tweakable split-domain cipher in CGO. Inputs and outputs to the encipher algorithm are shown in yellow, whereas the values shown in light brown correspond to the update functionality.

```
EU_K(T)
    $(J,S) \twoheadleftarrow K$
    $(\overline{K},\overline{T}) \leftarrow \mathsf{F}_S^1(T)$
    return $(\overline{K},\overline{T})$
```

```
EE_K^H(X_L, X_R)
    $(J,S) \twoheadleftarrow K$
    $Y_L \leftarrow \mathsf{E}_J^{H\|X_R}(X_L)$
    $Y_R \leftarrow \mathsf{F}_S^0(Y_L) \oplus X_R$
    return $(Y_L, Y_R)$
```

```
ED_K^H(Y_L, Y_R)
    $(J,S) \twoheadleftarrow K$
    $X_R \leftarrow \mathsf{F}_S^0(Y_L) \oplus Y_R$
    $X_L \leftarrow \mathsf{D}_J^{H\|X_R}(Y_L)$
    return $(X_L, X_R)$
```

Fig. 7. The encipher and decipher algorithms EE and ED comprising the UIV construction, augmented with an update functionality EU to form UIV+.

T and a single bit tweak. If the bit value is 0, then F returns a random string of size $|X_R|$, whereas if the input bit is 1, it returns a new state. Thus the input bit to F is set to 1 only when invoking the update functionality and is otherwise set to 0.

Again, the two subcomponents E and F can be instantiated in various ways, which will affect both the performance and concrete security of CGO. We will propose a concrete instantion for CGO shortly, and see Section 5 for fully instantiated CGO benchmarked against Tor's current onion encryption scheme.

The CRND$_\ell$ Security of UIV+. Degabriele and Karadžić already showed that their new construction UIV is RRND secure even though it fails as strong PRP. We leverage their result and prove that our augmented UIV+ construction is URRND and hence CRND$_\ell$ secure, stated in terms of the STPRP security of E and PRF security of F.

Theorem 1 (CRND$_\ell$ **Security of** UIV+)**.** *Let* E, F *be a tweakable block cipher and a pseudorandom function, respectively, and let* UIV+ *be the updatable tweakable split-domain cipher given in Fig. 7. Then, for any $\ell > 0$ there exist simple fully black box reductions $\mathbb{B}$ and $\mathbb{C}$ such that for all* CRND$_\ell$ *adversaries $\mathbb{A}$ against* UIV+

$$\mathsf{Adv}_{\mathsf{UIV+}}^{\mathrm{CRND}_\ell}(\mathbb{A}) \leq \ell \cdot q_{\mathrm{Up}} \Big(\mathsf{Adv}_{\mathsf{E}}^{\mathrm{stprp}}(\mathbb{B}^{\mathbb{A}}) + 2 \cdot \mathsf{Adv}_{\mathsf{F}}^{\mathrm{PRF}}(\mathbb{C}^{\mathbb{A}}) + \frac{v \cdot q_{\mathrm{Gu}}}{2^{n-1}} + \frac{q_1(q_1-1)}{2^{n+1}} + \frac{q_{\mathrm{En}}(q_{\mathrm{En}}-1)}{2^{n+1}} \Big),$$

where $\mathbb{B}$ and $\mathbb{C}$'s overheads are essentially linear in the number of queries by $\mathbb{A}$ and q_{Up} is the number of queries by the adversary $\mathbb{A}$ to the oracle Up.

Proof (Sketch). Degabriele and Karadžić [13] prove that the UIV construction is RRND secure:

$$\mathsf{Adv}_{\mathsf{UIV}}^{\mathrm{RRND}}(\mathbb{A}) \leq \mathsf{Adv}_{\mathsf{E}}^{\mathrm{stprp}}(\mathbb{B}^{\mathbb{A}}) + \mathsf{Adv}_{\mathsf{F}}^{\mathrm{PRF}}(\mathbb{C}^{\mathbb{A}}) + \frac{v \cdot q_{\mathrm{Gu}}}{2^{n-1}} + \frac{q_1(q_1-1)}{2^{n+1}} + \frac{q_{\mathrm{En}}(q_{\mathrm{En}}-1)}{2^{n+1}},$$

where $v > 0$, $\mathbb{A}$ makes $q_{\mathcal{O}}$ queries $\forall \mathcal{O} \in \{\mathrm{En}, \mathrm{De}, \mathrm{Gu}\}$, $v \cdot q_{\mathrm{Gu}} \leq 2^{n-1}$ and $q_1 = q_{\mathrm{En}} + q_{\mathrm{De}} + q_{\mathrm{Gu}}$.

We show that combining security for the update and RRND security grants URRND security and, in turn, that URRND implies CRND$_\ell$ security. The detailed proof is presented in Appendix A. We obtain the following chain of inequalities:

$$\begin{aligned}
\mathsf{Adv}^{\mathrm{CRND}_\ell}_{\mathsf{UIV+}}(\mathbb{A}) &\leq \ell \cdot q_{\mathrm{Up}} \cdot \mathsf{Adv}^{\mathrm{URRND}}_{\mathsf{UIV+}}(\mathbb{B}^{\mathbb{A}}) \\
&\leq \ell \cdot q_{\mathrm{Up}} \left(\mathsf{Adv}^{\mathrm{RRND}}_{\mathsf{UIV}}(\mathbb{B}^{\mathbb{A}}) + \mathsf{Adv}^{\mathrm{UPDT}}_{\mathsf{UIV+}}(\mathbb{C}^{\mathbb{A}}) \right) \\
&\leq \ell \cdot q_{\mathrm{Up}} \left(\mathsf{Adv}^{\mathrm{stprp}}_{\mathsf{E}}(\mathbb{B}^{\mathbb{A}}) + 2 \cdot \mathsf{Adv}^{\mathrm{PRF}}_{\mathsf{F}}(\mathbb{C}^{\mathbb{A}}) + \frac{v \cdot q_{\mathrm{Gu}}}{2^{n-1}} \right. \\
&\qquad \left. + \frac{q_1(q_1 - 1)}{2^{n+1}} + \frac{q_{\mathrm{En}}(q_{\mathrm{En}} - 1)}{2^{n+1}} \right).
\end{aligned}$$

GCM-UIV+: An Efficient Instantiation. In CGO, we instantiate UIV+ using GCM components [22, 35] in order to take advantage of x86 native instruction sets; accordingly, we call this concrete instantiation GCM-UIV+. Specifically, we realize the tweakable blockcipher E and the expanding F shown in Fig. 6 using separate instances of AES and POLYVAL. For AES, we concentrate below on the 128-bit key version, though we will address 256-bit keys later on. For POLYVAL [23], we recall that it has a 128-bit key and hashes any sequence of 128-bit strings to a single 128-bit string.

The tweakable blockcipher is instantiated through the LRW2 [31] construction with POLYVAL as the almost-XOR-universal hash function and AES for the blockcipher (see Fig. 8). The size of the left domain $n = |X_L|$ in GCM-UIV+ is fixed to 128 bits, namely the block size of the tweakable blockcipher inherited from AES, while the size of the right domain is fixed to $m = 493 \cdot 8$, where $493 = 509 - 16$ (the fixed Tor ciphertext length minus the bytes needed for the left domain). The overall key size (of J) is 256 bits. Although, without additional length encoding applied to the input, POLYVAL is only secure as an almost-XOR-universal hash function over strings of some fixed size, in Tor the sizes of ciphertexts and headers are luckily fixed. In GCM-UIV+ specifically, the sizes of X_R and H are fixed to 493 and 17 bytes (16 for the tag T and 1 for the associated data), respectively.

We realise the tweakable pseudorandom function F via a Hash-then-PRF composition using AES in counter mode and POLYVAL (see Fig. 8). For $b = 0$, F needs to output a string of size equal to $|X_R|$, which is fixed to 493 bytes. Thus, 31 'counts' suffice, as they produce $31 \cdot 128$ bits (equal to 496 bytes). For $b = 1$, the output consists of 128 bits to refresh the tag T (the size of $|X_L|$), a fresh key for the tweakable blockcipher (256 bits) and a fresh key for F itself (also 256 bits, namely 128 for AES and 128 for POLYVAL). Thus 5 'counts' suffice. To ensure domain separation between $b = 0$ and $b = 1$, we need 36 counts in total, which can be achieved with a 6-bit counter.

Using AES in counter mode, we could easily create a pseudorandom function mapping 122 bits to $36 \cdot 128$ bits, by appending the 122-bit input with a 6-bit counter. However, in the UIV+ construction, the input size of F must match the block size of E, namely 128 bits. By composing the above counter-mode PRF with a universal hash mapping 128 bits to 122 bits, we obtain a pseudorandom function with the required input size. For the universal hash we use POLYVAL and truncate the most significant 6 bits of its output.

Alternative instantiations. The description above immediately reveals an alternative based on AES with 256-bit keys. In that case, both the tweakable blockcipher and the tweakable pseudorandom function will instead use a 256-bit key and a (still) 128-bit POLYVAL key. For the $b = 1$ branch of the tweakable pseudorandom function, it means 7 'counts' are needed instead of 5 to refresh the key material. Luckily, with our 6-bit counter this increased count is not an issue.

Furthermore, we designed CGO and UIV+ such that an evaluation of $\mathsf{F}^1_S(T)$ is always preceded by a corresponding call to $\mathsf{F}^0_S(Y_L)$ with $T = Y_L$. We could exploit this clever circumstance by treating F_S as an extendable output function that can first output its bits for the 0-tweak branch, followed by whatever is needed for the 1-tweak branch (cf. the amortization for forkciphers [3]). Thus, replacing F_S with for instance SHAKE [18] is relatively straightforward.

Security of GCM-UIV+. The exact security of the above instantiations of E and F in GCM-UIV+ follows easily from known results in the literature. For the security of E, we exploit that it fits the LRW2 construction [31], so to obtain an exact bound, we only have to determine the universality parameter ϵ for our specific use of POLYVAL. In general, $\epsilon = \frac{\ell}{2^{128}}$, where ℓ is the message size in 128-bit blocks [22].

```
$\mathsf{F}_S^b(T)$
  $(L, B) \twoheadleftarrow S$, $Z = \varepsilon$
  if $b = 0$
    for $i = 0$ to $30$
      $Z \leftarrow Z \parallel \mathsf{AES}_L(\lfloor \mathsf{POLYVAL}(B, T) \rfloor_{122} \parallel i)$
  else
    for $i = 31$ to $35$
      $Z \leftarrow Z \parallel \mathsf{AES}_L(\lfloor \mathsf{POLYVAL}(B, T) \rfloor_{122} \parallel i)$
  return $Z$
```

```
$\mathsf{E}_J^{H \parallel X_R}(X_L)$
  $(\overline{L}, \overline{B}) \twoheadleftarrow J$
  $Z \leftarrow \mathsf{POLYVAL}(\overline{B}, H \parallel X_R)$
  $Y_L \leftarrow Z \oplus \mathsf{AES}_{\overline{L}}(Z \oplus X_L)$
  return $Y_L$

$\mathsf{D}_J^{H \parallel X_R}(Y_L)$
  $(\overline{L}, \overline{B}) \twoheadleftarrow J$
  $Z \leftarrow \mathsf{POLYVAL}(\overline{B}, H \parallel X_R)$
  $X_L \leftarrow Z \oplus \mathsf{AES}_{\overline{L}}^{-1}(Z \oplus Y_L)$
  return $X_L$
```

Fig. 8. Concrete instantiation of the UIV+ components using AES and POLYVAL, thereby giving rise to the GCM-UIV+ updatable split-domain cipher construction.

In our case, ℓ amounts to $493 + 3$ bytes, resulting in ϵ being roughly equal to 2^{-123}. The security of E is stated formally in Proposition 1.

Proposition 1 (Security of E (LRW2 cf. Theorem 2 [31])).
Let E be the tweakable blockcipher construction described in Fig. 8 composed from 128-bit AES and an 2^{-123}-AXU hash function. Then for any STPRP adversary $\mathbb{A}_{stprp}$ making q queries, there exists an SPRP adversary $\mathbb{A}_{sprp}$ making an equal number of queries, such that:

$$\mathsf{Adv}_{\mathsf{E}}^{\mathrm{STPRP}}(\mathbb{A}) \leq \mathsf{Adv}_{\mathsf{AES}}^{\mathrm{SPRP}}(\mathbb{A}) + \frac{3q^2}{2^{124}}\,.$$

As noted above, F follows the well-known Hash-then-PRF paradigm, where the underlying PRF consists of AES in counter mode and the universal hash is truncated POLYVAL. A security analysis of this composition in terms of the security of the underlying pseudorandom function and universal hash is known [29], reproduced in adapted form in Proposition 2. Here we assume, without loss of generality, that for every query the adversary receives outputs for both tweaks $b = 0$ and $b = 1$.

Proposition 2 (Security of F (Hash-then-PRF cf. Lemma 5.1 [29])). *Let F be the tweakable pseudorandom function construction described in Fig. 8 composed from counter-mode AES and an ϵ-AU hash function mapping 128-bit strings to 122-bit strings. Then for any PRF adversary $\mathbb{A}_{prf}$ against F making q queries, there exists a PRF adversary $\mathbb{A}_{ctr}$ against AES-CTR making q queries, such that:*

$$\mathsf{Adv}_{\mathsf{F}}^{\mathrm{PRF}}(\mathbb{A}_{\mathsf{F}}) \leq \mathsf{Adv}_{\mathsf{AES\text{-}CTR}}^{\mathrm{PRF}}(\mathbb{A}_{ctr}) + \frac{q^2}{2}\epsilon\,.$$

As any ϵ-AXU hash is automatically an ϵ-AU hash and the effect of truncation on these functions is well understood [12, Proposition 1], the quantitative security of truncated POLYVAL over 128-bit inputs works out to be $\epsilon = \frac{2^7}{2^{128}}$. On the other hand, the security of counter mode AES as a pseudorandom function follows easily from the PRP security of AES and the switching lemma [7,40], restated in adapted form in the following proposition.

Proposition 3 (Security of AES-CTR). *For any PRF adversary $\mathbb{A}_{prf}$ against AES-CTR making q queries, there exists a PRP adversary $\mathbb{A}_{prp}$ against AES making $36q$ queries, such that:*

$$\mathsf{Adv}_{\mathsf{AES\text{-}CTR}}^{\mathrm{PRF}}(\mathbb{A}_{prf}) \leq \mathsf{Adv}_{\mathsf{AES}}^{\mathrm{PRP}}(\mathbb{A}_{prp}) + \frac{(36q)^2}{2^{128+1}}\,.$$

Combining Proposition 2 and Proposition 3, and substituting for $\epsilon = 2^{-121}$ yields the exact security of F.

5 Implementation and Benchmarking

We report on our experimental results, where we measured the performance of our CGO implementation and compare it to the scheme currently deployed by Tor, which henceforth we refer to as Classic Tor.

C implementation. The Classic Tor implementation for our benchmarks uses the Crypto++ library [10] version 8.2 for SHA-1 processing due to its availability within SUPERCOP [9] and an intrinsic-driven C++ implementation for the CTR-mode and plain C++ for the remaining driver code. The benchmarked CGO implementation also uses Crypto++ though primarily for its storage and comparison facilities, the core functions are all implemented using AES-NI and PCLMUL intrinsics. These implementations use the optimized techniques by Gueron et al. [24, 25] for fast AES re-keying and fast POLYVAL reductions. Due to the intrinsic driven nature of the code, we had to decide on a primary target platform, for which we chose Intel's Skylake. We have not made use of AVX or AVX-512, but we expect performance to improve from the use of these additional instruction set extensions, potentially using the techniques from Gueron et al. [17]. Finally, while Crypto++ does internally use the SHA-NI hardware acceleration extension to accelerate SHA-1, support for this instruction set is relatively less common (especially for legacy systems) and accordingly we did not use it in the Classic Tor implementation.

Benchmarking methodology. In our benchmarks we made use of SUPERCOP [9] version 20200525 where we added a new primitive type for onion encryption in Tor that we called `crypto_tor`. We chose SUPERCOP for its flexibility and its ability to measure performance in CPU cycles accurately. We benchmark CGO both in the the forward and backward directions. For each direction we evaluate the performance of the cryptographic computation at the proxy, an intermediate router operating as a relay, and an end-point router either sending or receiving messages. Although the forward processing by a router is in theory modelled as a single algorithm (OR-Dec), the actual cryptographic processing performed by a router differs depending on whether it serves as a relay or exit node. This dichotomy is a consequence of Tor's 'leaky pipes' functionality and end-to-end integrity protection, and thus affects both CGO and Classic Tor.

One discrepancy between Classic Tor and CGO is the reduced payload for the latter (498 bytes versus 488 bytes, respectively). We account for this discrepancy by reporting cycles per byte of message. For each direction, we process a message of some given length and measure the CPU cycles needed for each operation. We conducted this benchmark for message lengths ranging from 200 to 5,000 bytes at 10-byte increments and recorded the median measurement over 100 iterations for each message length.

We obtained our benchmarks on an Intel Cascade Lake processor on Amazon's AWS cloud: `c5.metal`. We used Amazon's Linux distribution along with SUPERCOP version 20200525, GCC 7.3.1 and clang 11.1.0, with simultaneous multi-threading and Intel Turbo Boost turned off.

Results. The plots from our benchmarks are displayed in Fig. 9. In both directions, we observe that CGO yields a significant performance improvement over Classic Tor, roughly by a factor of three, in the cell processing done at the proxy and at an exit or entry router. In contrast, we observe a milder slowdown of around 20% to 50% in the processing done at the intermediate routers. Interestingly, this slowdown appears to be more pronounced in the backward direction. This slowdown at the intermediate routers is expected as here Classic Tor simply performs counter-mode encryption, which is hard to outperform.

Overall, the mild performance penalty in the intermediate routers is offset by considerably larger performance improvements at exit and entry routerr. These performance improvements are significant since exit and entry routers are a scarce resource in the Tor network due to their increased exposure and liability; moreover, performance at the proxy is particularly critical when running a Tor client on a mobile phone or another lightweight device. Furthermore, CGO attains much better overall security than Classic Tor.

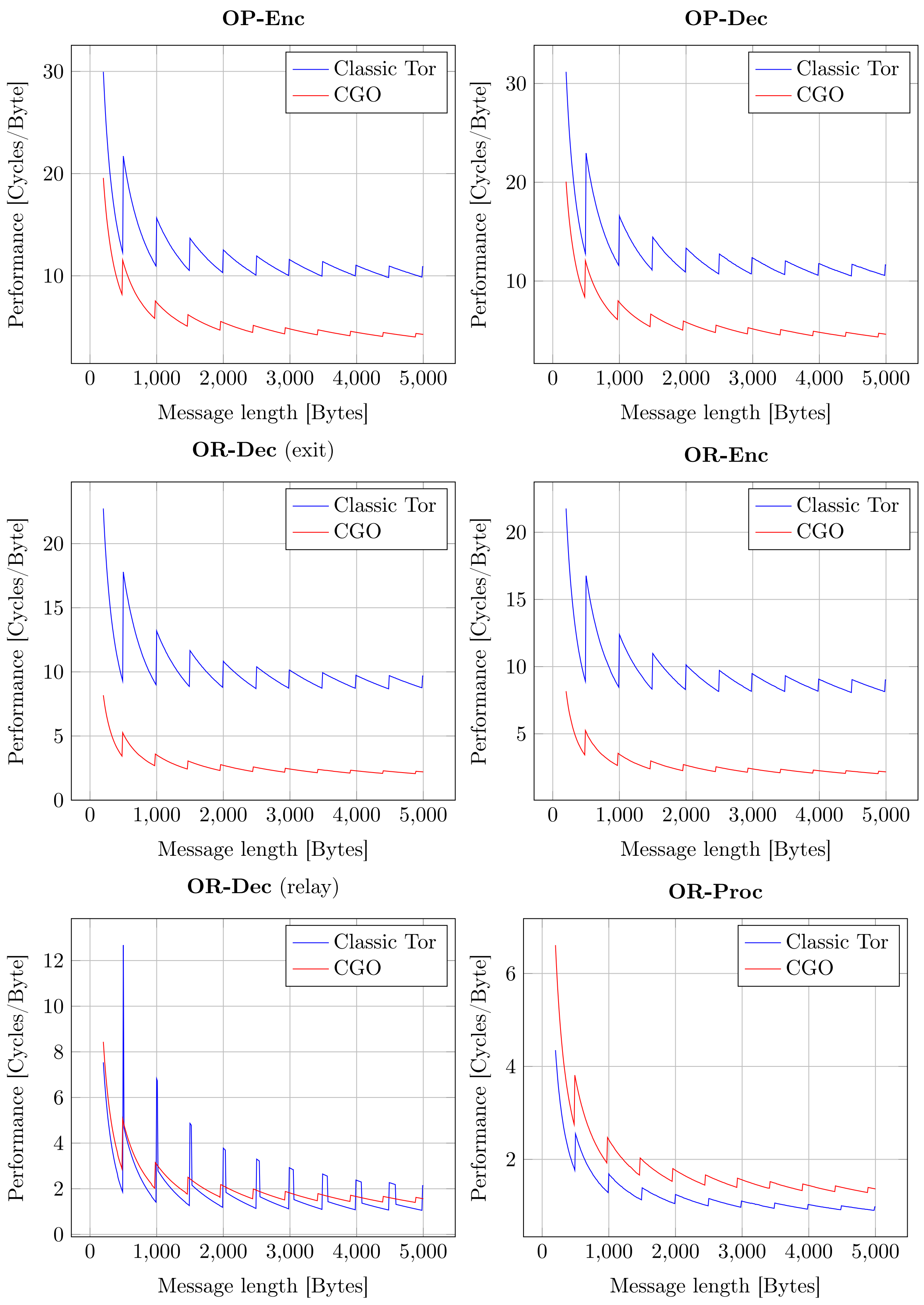

Fig. 9. Performance comparison between Classic Tor and CGO on an Intel Cascade Lake processor, showing the median CPU cycles per byte for varying message lengths. The plots on the left correspond to the forward direction and on the right side is the backward direction.

References

1. Total relay bandwidth in Tor. https://metrics.torproject.org/bandwidth.html?start=2000-01-01&end=2024-06-12 (June 2024)
2. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: How to securely release unverified plaintext in authenticated encryption. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 105–125. Springer, Berlin, Heidelberg (Dec 2014). https://doi.org/10.1007/978-3-662-45611-8_6
3. Andreeva, E., Lallemand, V., Purnal, A., Reyhanitabar, R., Roy, A., Vizár, D.: Forkcipher: A new primitive for authenticated encryption of very short messages. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part II. LNCS, vol. 11922, pp. 153–182. Springer, Cham (Dec 2019). https://doi.org/10.1007/978-3-030-34621-8_6
4. Ashur, T., Dunkelman, O., Luykx, A.: Boosting authenticated encryption robustness with minimal modifications. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 3–33. Springer, Cham (Aug 2017). https://doi.org/10.1007/978-3-319-63697-9_1
5. Baecher, P., Brzuska, C., Fischlin, M.: Notions of black-box reductions, revisited. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 296–315. Springer, Berlin, Heidelberg (Dec 2013). https://doi.org/10.1007/978-3-642-42033-7_16
6. Barwell, G., Page, D., Stam, M.: Rogue decryption failures: Reconciling AE robustness notions. In: Groth, J. (ed.) 15th IMA International Conference on Cryptography and Coding. LNCS, vol. 9496, pp. 94–111. Springer, Cham (Dec 2015). https://doi.org/10.1007/978-3-319-27239-9_6
7. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: 38th FOCS. pp. 394–403. IEEE Computer Society Press (Oct 1997). https://doi.org/10.1109/SFCS.1997.646128
8. Bellare, M., Rogaway, P.: Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 317–330. Springer, Berlin, Heidelberg (Dec 2000). https://doi.org/10.1007/3-540-44448-3_24
9. Bernstein, D.J., (editors), T.L.: eBACS: ECRYPT Benchmarking of Cryptographic Systems. https://bench.cr.yp.to, accessed 5 May 2020
10. Dai, W., Walton, J., Contributors: Crypto++: free C++ Class Library of Cryptographic Schemes. https://cryptopp.com/
11. Danezis, G., Goldberg, I.: Sphinx: A compact and provably secure mix format. In: 2009 IEEE Symposium on Security and Privacy. pp. 269–282. IEEE Computer Society Press (May 2009). https://doi.org/10.1109/SP.2009.15
12. Degabriele, J.P., Gilcher, J., Govinden, J., Paterson, K.G.: SoK: Efficient design and implementation of polynomial hash functions over prime fields. In: 2024 IEEE Symposium on Security and Privacy. pp. 3128–3146. IEEE Computer Society Press (May 2024). https://doi.org/10.1109/SP54263.2024.00132
13. Degabriele, J.P., Karadžić, V.: Overloading the nonce: Rugged PRPs, nonce-set AEAD, and order-resilient channels. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 264–295. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15985-5_10
14. Degabriele, J.P., Karadžić, V.: Populating the zoo of rugged pseudorandom permutations. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Part VIII. LNCS, vol. 14445, pp. 270–300. Springer, Singapore (Dec 2023). https://doi.org/10.1007/978-981-99-8742-9_9
15. Degabriele, J.P., Stam, M.: Untagging Tor: A formal treatment of onion encryption. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 259–293. Springer, Cham (Apr / May 2018). https://doi.org/10.1007/978-3-319-78372-7_9
16. Dingledine, R., Mathewson, N., Syverson, P.F.: Tor: The second-generation onion router. In: USENIX Security Symposium. pp. 303–320. USENIX (2004)
17. Drucker, N., Gueron, S., Krasnov, V.: Making AES great again: the forthcoming vectorized AES instruction. Cryptology ePrint Archive, Report 2018/392 (2018), https://eprint.iacr.org/2018/392
18. Dworkin, M.J.: SHA-3 standard: Permutation-based hash and extendable-output functions (2015). https://doi.org/https://doi.org/10.6028/NIST.FIPS.202
19. Fu, X., Ling, Z., Luo, J., Yu, W., Jia, W., Zhao, W.: One cell is enough to break tor's anonymity. In: Proceedings of Black Hat Technical Security Conference. pp. 578–589. Citeseer (2009)
20. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions (extended abstract). In: 25th FOCS. pp. 464–479. IEEE Computer Society Press (Oct 1984). https://doi.org/10.1109/SFCS.1984.715949
21. Goldschlag, D.M., Reed, M.G., Syverson, P.F.: Hiding routing information. In: Anderson, R.J. (ed.) Information Hiding, First International Workshop, Cambridge, UK, May 30 - June 1, 1996, Proceedings. Lecture Notes in Computer Science, vol. 1174, pp. 137–150. Springer (1996). https://doi.org/10.1007/3-540-61996-8_37, https://doi.org/10.1007/3-540-61996-8_37
22. Gueron, S., Langley, A., Lindell, Y.: AES-GCM-SIV: Specification and analysis. Cryptology ePrint Archive, Report 2017/168 (2017), https://eprint.iacr.org/2017/168

23. Gueron, S., Langley, A., Lindell, Y.: AES-GCM-SIV: Nonce misuse-resistant authenticated encryption. RFC 8452 (Apr 2019). https://doi.org/10.17487/RFC8452, https://www.rfc-editor.org/info/rfc8452
24. Gueron, S., Lindell, Y.: GCM-SIV: Full nonce misuse-resistant authenticated encryption at under one cycle per byte. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 109–119. ACM Press (Oct 2015). https://doi.org/10.1145/2810103.2813613
25. Gueron, S., Lindell, Y., Nof, A., Pinkas, B.: Fast garbling of circuits under standard assumptions. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 567–578. ACM Press (Oct 2015). https://doi.org/10.1145/2810103.2813619
26. Halevi, S., Rogaway, P.: A parallelizable enciphering mode. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 292–304. Springer, Berlin, Heidelberg (Feb 2004). https://doi.org/10.1007/978-3-540-24660-2_23
27. Hern, A.: US defence department funded carnegie mellon research to break Tor. https://www.theguardian.com/technology/2016/feb/25/us-defence-department-funding-carnegie-mellon-research-break-tor (February 2016)
28. Hoang, V.T., Krovetz, T., Rogaway, P.: Robust authenticated-encryption AEZ and the problem that it solves. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 15–44. Springer, Berlin, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_2
29. Jha, A., Nandi, M.: A survey on applications of H-technique: Revisiting security analysis of PRP and PRF. Cryptology ePrint Archive, Report 2018/1130 (2018), https://eprint.iacr.org/2018/1130
30. Lewko, A.B., Waters, B.: Why proving HIBE systems secure is difficult. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 58–76. Springer, Berlin, Heidelberg (May 2014). https://doi.org/10.1007/978-3-642-55220-5_4
31. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 31–46. Springer, Berlin, Heidelberg (Aug 2002). https://doi.org/10.1007/3-540-45708-9_3
32. Luby, M., Rackoff, C.: How to construct pseudo-random permutations from pseudo-random functions (abstract). In: Williams, H.C. (ed.) CRYPTO'85. LNCS, vol. 218, p. 447. Springer, Berlin, Heidelberg (Aug 1986). https://doi.org/10.1007/3-540-39799-X_34
33. Mathewson, N.: Proposal 202: Two improved relay encryption protocols for Tor cells. https://github.com/torproject/torspec/blob/main/proposals/202-improved-relay-crypto.txt (June 2012)
34. Mathewson, N.: Proposal 261: AEZ for relay cryptography. https://github.com/torproject/torspec/blob/main/proposals/261-aez-crypto.txt (October 2015)
35. McGrew, D.A., Viega, J.: The security and performance of the Galois/counter mode (GCM) of operation. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 343–355. Springer, Berlin, Heidelberg (Dec 2004). https://doi.org/10.1007/978-3-540-30556-9_27
36. Raccoon, T.: How i learned to stop ph34ring nsa and love the base rate fallacy. http://archives.seul.org/or/dev/Sep-2008/msg00016.html (September 2008)
37. Raccoon, T.: Analysis of the relative severity of tagging attacks. http://archives.seul.org/or/dev/Mar-2012/msg00019.html (March 2012)
38. Reed, M.G., Syverson, P.F., Goldschlag, D.M.: Proxies for anonymous routing. In: ACSAC'96. pp. 95–104. IEEE Computer Society (1996)
39. Reingold, O., Trevisan, L., Vadhan, S.P.: Notions of reducibility between cryptographic primitives. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 1–20. Springer, Berlin, Heidelberg (Feb 2004). https://doi.org/10.1007/978-3-540-24638-1_1
40. Rogaway, P.: Evaluation of some blockcipher modes of operation. Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan **630** (2011)
41. Rogaway, P., Zhang, Y.: Onion-AE: Foundations of nested encryption. PoPETs **2018**(2), 85–104 (Apr 2018). https://doi.org/10.1515/popets-2018-0014
42. arma (Roger Dingledine): Tor security advisory: "relay early" traffic confirmation attack. https://blog.torproject.org/blog/tor-security-advisory-relay-early-traffic-confirmation-attack (July 2014)
43. Shrimpton, T., Terashima, R.S.: A modular framework for building variable-input-length tweakable ciphers. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 405–423. Springer, Berlin, Heidelberg (Dec 2013). https://doi.org/10.1007/978-3-642-42033-7_21
44. Syverson, P., Tsudik, G., Reed, M., Landwehr, C.: Towards an analysis of onion routing security. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings. pp. 96–114. Springer Berlin Heidelberg, Berlin, Heidelberg (2001). https://doi.org/10.1007/3-540-44702-4_6, https://doi.org/10.1007/3-540-44702-4_6
45. Syverson, P.F., Goldschlag, D.M., Reed, M.G.: Anonymous connections and onion routing. In: 1997 IEEE Symposium on Security and Privacy. pp. 44–54. IEEE Computer Society Press (1997). https://doi.org/10.1109/SECPRI.1997.601314
46. Tomer Ashur, Orr Dunkelman, A.L.: Using ADL for relay cryptography (solving the crypto-tagging-attack). https://github.com/torproject/torspec/blob/dc3683f929a747876cfb66fa1b91edff68dfc27d/proposals/295-relay-crypto-with-adl.txt (July 2019)

A The Proof of Theorem 1

The main goal is to bound the advantage $\mathsf{Adv}^{\mathrm{CRND}_\ell}_{\mathsf{UIV+}}(\mathbb{A})$ in terms of $\mathsf{Adv}^{\mathrm{RRND}}_{\mathsf{UIV}}(\mathbb{A})$ and the PRF security of F. This, combined with results by Degabriele and Karadžić, completes the proof of Theorem 1. In the first section, we analyse the relations between the security notions RRND and URRND; the second part shows how URRND and CRND$_\ell$ relate. Finally, we bound the security of the update functionality in terms of the PRF security of F.

A.1 Decomposing URRND Security into RRND and UPDT

We prove that the URRND security of an updatable tweakable split-domain cipher $(\mathsf{EE}, \mathsf{EU})$ is equivalent to the RRND security of EE together with the UPDT security of EU. To do so, we redefine the experiment $\mathsf{Exp}^{\mathrm{URRND}\text{-}b}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A})$ to use two separate bits instead of one, namely b_{Up} and b_{RRND}, as shown in Fig. 10. This way we implicitly define two hybrid experiments yielding two additional security notions: one which corresponds to the RRND game and the other we call UPDT. Bit b_{Up} controls the behaviour of the Up oracle, while bit b_{RRND} controls the behaviour of the En, De and Gu oracles. In particular:

- when $b_{\mathrm{Up}} = b_{\mathrm{RRND}}$ we recover the experiments corresponding to the URRND security of $(\mathsf{EE}, \mathsf{EU})$;
- when $b_{\mathrm{RRND}} = 1$ and b_{Up} is either 0 or 1, the two experiments define UPDT security for $(\mathsf{EE}, \mathsf{EU})$;
- when $b_{\mathrm{Up}} = 0$ and b_{RRND} is either 0 or 1, we recover an analogue of RRND security with respect to an *updatable* tweakable split-domain cipher.

This is stated more formally below, where we state the advantages in terms of the two-bit experiments.

Definition 3 (URRND, UPDT, RRND **advantages).** *For any split-domain tweakable cipher* EE *over* $\mathcal{D}_L \times \mathcal{D}_R$ *and any associated update function* EU*, the corresponding* URRND*,* UPDT*, and* RRND *advantages of an adversary* $\mathbb{A}$ *are given by:*

$$\mathsf{Adv}^{\mathrm{URRND}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = \Pr\Big[\mathsf{Exp}^{\mathrm{URRND}\text{-}0,0}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big] - \Pr\Big[\mathsf{Exp}^{\mathrm{URRND}\text{-}1,1}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big].$$

$$\mathsf{Adv}^{\mathrm{UPDT}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = \Pr\Big[\mathsf{Exp}^{\mathrm{URRND}\text{-}0,1}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big] - \Pr\Big[\mathsf{Exp}^{\mathrm{URRND}\text{-}1,1}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big].$$

$$\mathsf{Adv}^{\mathrm{RRND}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = \Pr\Big[\mathsf{Exp}^{\mathrm{URRND}\text{-}0,0}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big] - \Pr\Big[\mathsf{Exp}^{\mathrm{URRND}\text{-}0,1}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big].$$

where $\mathsf{Exp}^{\mathrm{URRND}\text{-}b_{\mathrm{Up}},b_{\mathrm{RRND}}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A})$ *is defined in Fig. 10.*

By inspection, the new definition of $\mathsf{Adv}^{\mathrm{URRND}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A})$ is equivalent to its previous definition, whereas $\mathsf{Adv}^{\mathrm{RRND}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A})$ is effectively a reformulation of Definition 1, where the adversary additionally has access to a random update oracle that conveys no additional capabilities. For convenience, whenever $\mathbb{A}$ is an adversary without access to an update oracle, $\mathbb{A}^+$ denotes the adversary that has access to an update oracle but simply does not call it. Conversely, whenever $\mathbb{A}$ is an adversary with access to an update oracle, $\mathbb{A}^-$ denotes an adversary without such access but instead internally answers all update queries with fresh randomness. Technically, both $\mathbb{A}^+$ and $\mathbb{A}^-$ are simple, fully black-box reductions [5, 30, 39] that, aside from $\mathbb{A}^-$'s randomness generation, are as efficient and effective as their respective adversary $\mathbb{A}$.

The main result of this section is Lemma 1, bounding URRND security in terms of RRND and UPDT security. On the other hand, Lemma 2 and Proposition 4 are secondary results, included solely for completeness.

Lemma 1 (RRND $\wedge$ UPDT **security** $\implies$ URRND **security).** *Let* $(\mathsf{EE}, \mathsf{EU})$ *be an updatable tweakable split-domain cipher and* $\mathbb{A}$ *an* URRND *adversary* $\mathbb{A}$ *against* $(\mathsf{EE}, \mathsf{EU})$*. Then*

$$\mathsf{Adv}^{\mathrm{URRND}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) \le \mathsf{Adv}^{\mathrm{UPDT}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) + \mathsf{Adv}^{\mathrm{RRND}}_{\mathsf{EE}}(\mathbb{A}^-)\ .$$

Proof. The result comes straightforward from the definition:

$$\begin{aligned}
\mathsf{Adv}^{\mathrm{URRND}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) &= \Pr\Big[\mathsf{Exp}^{\mathrm{URRND}\text{-}0,0}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big] - \Pr\Big[\mathsf{Exp}^{\mathrm{URRND}\text{-}1,1}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big] \\
&= \Pr\Big[\mathsf{Exp}^{\mathrm{URRND}\text{-}0,0}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big] - \Pr\Big[\mathsf{Exp}^{\mathrm{URRND}\text{-}0,1}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big] \\
&\qquad + \Pr\Big[\mathsf{Exp}^{\mathrm{URRND}\text{-}0,1}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big] - \Pr\Big[\mathsf{Exp}^{\mathrm{URRND}\text{-}1,1}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big] \\
&= \mathsf{Adv}^{\mathrm{UPDT}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) + \mathsf{Adv}^{\mathrm{RRND}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) \\
&\le \mathsf{Adv}^{\mathrm{UPDT}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) + \mathsf{Adv}^{\mathrm{RRND}}_{\mathsf{EE}}(\mathbb{A}^-)\ .
\end{aligned}$$

```
Experiment $\mathsf{Exp}^{\text{URRND-}b_{\text{Up}},b_{\text{RRND}}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A})$
    $K \leftarrow\!\!\$\ \{0,1\}^k$
    $\hat{b} \leftarrow \mathbb{A}^{\text{En},\text{De},\text{Gu},\text{Up}}$
    return $\hat{b}$

En$(H, X_L, X_R)$
    if $b_{\text{RRND}} = 1$
        $(Y_L, Y_R) \leftarrow \mathsf{EE}^H_K(X_L, X_R)$
    else
        $(Y_L, Y_R) \leftarrow\!\!\$\ \mathcal{D}_L \times \mathcal{D}_R$
    $\mathcal{F} \xleftarrow{\cup} Y_L,\ \mathcal{U} \xleftarrow{\cup} (H, Y_L, Y_R)$
    return $(Y_L, Y_R)$

Gu$(H, Y_L, Y_R, X'_L)$
    if $(H, Y_L, Y_R) \in \mathcal{U}$
        return $\lightning$
    if $b_{\text{RRND}} = 1$
        $(X_L, X_R) \leftarrow \mathsf{ED}^H_K(Y_L, Y_R)$
        return $X_L = X'_L$
    else
        return false

De$(H, Y_L, Y_R)$
    if $Y_L \in \mathcal{F}$
        return $\lightning$
    if $b_{\text{RRND}} = 1$
        $(X_L, X_R) \leftarrow \mathsf{ED}^H_K(Y_L, Y_R)$
    else
        $(X_L, X_R) \leftarrow\!\!\$\ \mathcal{D}_L \times \mathcal{D}_R$
    $\mathcal{F} \xleftarrow{\cup} Y_L,\ \mathcal{U} \xleftarrow{\cup} H, Y_L, Y_R)$
    return $(X_L, X_R)$

Up$(T)$
    if $b_{\text{Up}} = 1$
        $(\overline{K}, \overline{T}) \leftarrow \mathsf{EU}_K(T)$
    else
        $(\overline{K}, \overline{T}) \leftarrow\!\!\$\ \{0,1\}^k \times \mathcal{D}_L$
    return $(\overline{K}, \overline{T})$
```

Fig. 10. The $\mathsf{Exp}^{\text{URRND-}b_{\text{Up}},b_{\text{RRND}}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A})$ experiments used to define UPDT and equivalent formulations of URRND and RRND security for an updatable tweakable split-domain cipher.

□

Lemma 2 (URRND security $\Longrightarrow$ RRND security). *Let* $(\mathsf{EE}, \mathsf{EU})$ *be an updatable tweakable split-domain cipher and adv an* RRND *adversaries* $\mathbb{A}$ *against* EE*, then*

$$\mathsf{Adv}^{\text{RRND}}_{\mathsf{EE}}(\mathbb{A}) \leq \mathsf{Adv}^{\text{URRND}}_{(\mathsf{EE},\mathsf{EU})}(\mathbb{A}^+) .$$

Proof. Since $b_{\text{Up}}\ = 0$ in the RRND game, the oracle Up is completely independent from the other oracles, any secret, and the RPRP bit b_{RRND} , so $\mathbb{A}^+$'s simulation of its behaviour is perfect. For the other oracles $\mathbb{A}^+$, effectively forwards all queries from $\mathbb{A}$ to its oracles En, De, and Gu, returning the output bit obtained from $\mathbb{A}$, hence $\mathsf{Adv}^{\text{RRND}}_{\mathsf{EU}}(\mathbb{A}) \leq \mathsf{Adv}^{\text{URRND}}_{(\mathsf{EE},\mathsf{EU})}(\mathbb{A}^+)$. □

Proposition 4 (URRND security $\Longrightarrow$ UPDT security). *Let* $(\mathsf{EE}, \mathsf{EU})$ *be an updatable tweakable split-domain cipher and* $\mathbb{A}$ *an* UPDT *adversary against* EE*. Then*

$$\mathsf{Adv}^{\text{UPDT}}_{(\mathsf{EE},\mathsf{EU})}(\mathbb{A}) \leq 2 \cdot \mathsf{Adv}^{\text{URRND}}_{(\mathsf{EE},\mathsf{EU})}(\mathbb{A}) .$$

Proof. Again, by definition,

$$\begin{aligned}
\mathsf{Adv}^{\text{UPDT}}_{\mathsf{EE}}(\mathbb{A}) &= \Pr\Big[\mathsf{Exp}^{\text{URRND-0,1}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big] - \Pr\Big[\mathsf{Exp}^{\text{URRND-1,1}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big] \\
&= \Pr\Big[\mathsf{Exp}^{\text{URRND-0,1}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big] - \Pr\Big[\mathsf{Exp}^{\text{URRND-0,0}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big] \\
&\qquad + \Pr\Big[\mathsf{Exp}^{\text{URRND-0,0}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big] - \Pr\Big[\mathsf{Exp}^{\text{URRND-1,1}}_{\mathsf{EE},sdu}(\mathbb{A}) = 1\Big] \\
&= \mathsf{Adv}^{\text{RRND}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) + \mathsf{Adv}^{\text{URRND}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) \leq 2 \cdot \mathsf{Adv}^{\text{URRND}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}),
\end{aligned}$$

where the last inequality follows along the lines of Lemma 2. □

A.2 Introducing Corruption and Real Key Updates

In the security experiment $\mathsf{Exp}_{\mathsf{EE}}^{\text{URRND-}b_{\text{Up}},1}$, the adversary can query the oracle Up and, depending on the challenge bit b_{Up} , obtain either the output of $\mathsf{EU}_K(T)$ or an element of $\{0,1\}^k \times \mathcal{D}_L$ sampled uniformly at random. We can interpret this approach as a single key experiment in which the adversary can obtain what would become the next key and continue to query the oracles En, De, and Gu on the original key.

In Fig. 11, we extend the experiment to include real key updates by the oracle Up and corruption of the key material by the adversary. Essentially, the experiment is a cleaned up version of the CRND$_\ell$ game (Fig. 5) when $\ell = 1$. Thus, we allow the adversary to query the oracles En and De on previous keys after they have already been updated. The adversary queries oracles En and De on the jth key, and the experiment makes sure that enough keys have already been generated by verifying whether $j > i$, where the latter is used as counter of the queries to the oracle Up and, hence, represents the number of available keys.

The flag $fresh$ is used to verify whether the key is fresh and can be disclosed to the adversary: the key update by the oracle Up sets the flag to true, while the other oracles En, De, and Gu set it to false when queried on the most recent key. The set $\mathcal{Z}$ is initially empty, but contains the corrupted key handle 1 inherited from Fig. 5. Effectively, the adversary can obtain only the most recent key and corrupting it terminates useful access to all oracles.

Definition 4 (CRND **Advantage).** *For any tweakable split-domain cipher* EE, EU *over* $\mathcal{D}_L \times \mathcal{D}_R$ *and any update function* EU*, the corresponding* CRND *advantage of an adversary* $\mathbb{A}$ *is given by:*

$$\mathsf{Adv}_{\mathsf{EE},\mathsf{EU}}^{\text{CRND}}(\mathbb{A}) = \Pr\left[\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\text{CRND-1}}(\mathbb{A}) = 1\right] - \Pr\left[\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\text{CRND-0}}(\mathbb{A}) = 1\right],$$

where $\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\text{CRND-}b}(\mathbb{A})$ *is defined in Fig. 11.*

Hybrid Argument for CRND Security. We prove the construction secure even in case the real key update is performed, by hybrid argument over the number q_{Up} of queries to the oracle Up by the adversary. In each step of the hybrid argument we swap out the output of a single EU call for a random key. UPDT security is defined in terms of the original experiment $\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\text{URRND-}b_{\text{Up}},b_{\text{RRND}}}$ from Fig. 10, where the Up oracle returns the key to the adversary: this behaviour corresponds to a single Up query followed by a corruption, i.e., a query to the oracle Co. On the other hand, CRND security is defined in terms of the experiment $\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\text{CRND-}b}$ (Fig. 11), assuming the adversary performs q_{Up} queries to the oracle Up before querying the oracle Co.

Lemma 3 (URRND **security** $\implies$ CRND **security).** *Let* $(\mathsf{EE}, \mathsf{EU})$ *be an updatable split-domain tweakable cipher. Then there exists a simple fully black box reduction* $\mathbb{B}$ *such that, for all* CRND *adversaries* $\mathbb{A}$ *against* $(\mathsf{EE}, \mathsf{EU})$ *performing* q_{Up} *queries to the oracle* Up*,*

$$\mathsf{Adv}_{\mathsf{EE},\mathsf{EU}}^{\text{CRND}}(\mathbb{A}) \le q_{\text{Up}} \cdot \mathsf{Adv}_{\mathsf{EE},\mathsf{EU}}^{\text{URRND}}(\mathbb{B}^{\mathbb{A}}).$$

Proof. We prove the statement by a standard hybrid argument, and we define experiments $\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\mathsf{k}}$ (Fig. 12), where $0 \le k \le q_{\text{Up}}$ is the index of the last random key in experiment $\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\mathsf{k}}$ (the initial key $K_0 \leftarrow\$ \{0,1\}^k$ is always random). These experiments emulate the previous experiment $\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\text{CRND-}b}$, but they control the Up oracle according to the number of queries i instead of the challenge bit.

The original experiment $\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\text{CRND-}b}$ can be expressed in terms of $\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\mathsf{k}}$: the "random" experiment $\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\text{CRND-0}}$ corresponds to $\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{q_{\text{Up}}}$, since it is always true that $i \le q_{\text{Up}}$, and the "real" experiment $\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\text{CRND-1}}$ corresponds to $\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{0}$, since $i \ge 0$. We express the adversarial advantage $\mathsf{Adv}_{\mathsf{EE},\mathsf{EU}}^{\text{URRND}}(\mathbb{A})$ in terms of the experiments $\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\mathsf{k}}$:

$$\begin{aligned}
\mathsf{Adv}_{\mathsf{EE},\mathsf{EU}}^{\text{CRND}}(\mathbb{A}) &= \Pr\left[\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\text{CRND-1}}(\mathbb{A}) = 1\right] - \Pr\left[\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\text{CRND-0}}(\mathbb{A}) = 1\right] \\
&= \Pr\left[\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{0}(\mathbb{A}) = 1\right] - \Pr\left[\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{q_{\text{Up}}}(\mathbb{A}) = 1\right] = \sum_{k=1}^{q_{\text{Up}}} \left(\Pr\left[\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\mathsf{k-1}}(\mathbb{A}) = 1\right] - \Pr\left[\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\mathsf{k}}(\mathbb{A}) = 1\right]\right) \\
&\le q_{\text{Up}} \cdot \max_{k=1,\ldots,q_{\text{Up}}} \left(\Pr\left[\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\mathsf{k-1}}(\mathbb{A}) = 1\right] - \Pr\left[\mathsf{Exp}_{\mathsf{EE},\mathsf{EU}}^{\mathsf{k}}(\mathbb{A}) = 1\right]\right)
\end{aligned}$$

```
Experiment $\mathsf{Exp}^{\text{CRND-}b}_{\mathsf{EE},sdu}(\mathbb{A})$
  $K \leftarrow\!\!\$ \{0,1\}^k$
  $fresh \leftarrow \mathsf{true}$
  $i \leftarrow 0$
  $\hat{b} \leftarrow \mathbb{A}^{\text{En,De,Gu,Up,Co}}$
  **return** $\hat{b}$

En$(j, H, X_L, X_R)$
  **if** $(j > i) \vee (j = i \wedge \mathcal{Z} \neq \emptyset)$
    **return** $\lightning$
  **if** $j = i$
    $fresh \leftarrow \mathsf{false}$
  **if** $b = 1$
    $(Y_L, Y_R) \leftarrow \mathsf{EE}^H_{K_j}(X_L, X_R)$
  **else**
    $(Y_L, Y_R) \leftarrow\!\!\$ \mathcal{D}_L \times \mathcal{D}_R$
  $\mathcal{F}^j \overset{\cup}{\longleftarrow} Y_L,\ \mathcal{U}^j \overset{\cup}{\longleftarrow} (H, Y_L, Y_R)$
  **return** $(Y_L, Y_R)$

Gu$(j, H, Y_L, Y_R, X'_L)$
  **if** $(j > i) \vee (j = i \wedge \mathcal{Z} \neq \emptyset) \vee (H, Y_L, Y_R) \in \mathcal{U}^j$
    **return** $\lightning$
  **if** $j = i$
    $fresh \leftarrow \mathsf{false}$
  **if** $b = 1$
    $(X_L, X_R) \leftarrow \mathsf{ED}^H_{K_j}(Y_L, Y_R)$
    **return** $X_L = X'_L$
  **else**
    **return** $\mathsf{false}$

De$(j, H, Y_L, Y_R)$
  **if** $(j > i) \vee (j = i \wedge \mathcal{Z} \neq \emptyset) \vee (Y_L \in \mathcal{F}^j)$
    **return** $\lightning$
  **if** $j = i$
    $fresh \leftarrow \mathsf{false}$
  **if** $b = 1$
    $(X_L, X_R) \leftarrow \mathsf{ED}^H_{K_j}(Y_L, Y_R)$
  **else**
    $(X_L, X_R) \leftarrow\!\!\$ \mathcal{D}_L \times \mathcal{D}_R$
  $\mathcal{F}^j \overset{\cup}{\longleftarrow} Y_L,\ \mathcal{U}^j \overset{\cup}{\longleftarrow} (H, Y_L, Y_R)$
  **return** $(X_L, X_R)$

Up$(T)$
  **if** $\mathcal{Z} \neq \emptyset$
    **return** $\lightning$
  **if** $b = 1$
    $(K_{i+1}, T) \leftarrow \mathsf{EU}_{K_i}(T)$
  **else**
    $K_{i+1} \times T \leftarrow\!\!\$ \{0,1\}^k \times \mathcal{D}_L$
  $i \leftarrow i + 1, fresh \leftarrow \mathsf{true}$
  **return** $T$

Co()
  **if** $(\mathcal{Z} \neq \emptyset) \vee (fresh = \mathsf{false})$
    **return** $\lightning$
  $\mathcal{Z} \leftarrow \{1\}$
  **return** $K_i$
```

Fig. 11. The simplified experiment $\mathsf{Exp}^{\text{CRND-}b}_{\mathsf{EE,EU}}(\mathbb{A})$, where we introduce key corruption and real key updates. The oracle Co verifies whether a key has been used or not before revealing it to the adversary.

```
Experiment $\mathsf{Exp}^{\mathrm{k}}_{\mathsf{EE,EU}}(\mathbb{A})$
    $K_0 \leftarrow\!\!\$\ \{0,1\}^k$
    $fresh \leftarrow \mathsf{true}$
    $i \leftarrow 0$
    $\hat{b} \leftarrow \mathbb{A}^{\mathrm{En,De,Gu,Up,Co}}$
    return $\hat{b}$

Up($T$)
    if $\mathcal{Z} \neq \emptyset \vee i = q_{\mathrm{Up}}$ then return $\lightning$
    if $i \geq k$ then $(K_{i+1}, T) \leftarrow \mathsf{EU}_{K_i}(T)$
    else $(K_{i+1}, T) \leftarrow\!\!\$\ \{0,1\}^k \times \mathcal{D}_L$
    $i \leftarrow i + 1$
    $fresh \leftarrow \mathsf{true}$
    return $T$

Gu($j, H, Y_L, Y_R, X'_L$)
    if $j > i$ then return $\lightning$
    if $\mathcal{Z} \neq \emptyset \wedge j = i$ then return $\lightning$
    $(X_L, X_R) \leftarrow \mathsf{ED}^H_{K_j}(Y_L, Y_R)$
    $fresh \leftarrow \mathsf{false}$
    return $X_L = X'_L$

En($j, H, X_L, X_R$)
    if $j > i$ then return $\lightning$
    if $\mathcal{Z} \neq \emptyset \wedge j = i$ then return $\lightning$
    $(Y_L, Y_R) \leftarrow \mathsf{EE}^H_{K_j}(X_L, X_R)$
    $\mathcal{F}^j \xleftarrow{\cup} Y_L$
    $fresh \leftarrow \mathsf{false}$
    return $(Y_L, Y_R)$

De($j, H, Y_L, Y_R$)
    if $j > i$ then return $\lightning$
    if $\mathcal{Z} \neq \emptyset \wedge j = i$ then return $\lightning$
    if $Y_L \in \mathcal{F}^j$ then return $\lightning$
    $(X_L, X_R) \leftarrow \mathsf{ED}^H_{K_j}(Y_L, Y_R)$
    $\mathcal{F}^j \xleftarrow{\cup} Y_L$
    $fresh \leftarrow \mathsf{false}$
    return $(X_L, X_R)$

Co
    if $\mathcal{Z} \neq \emptyset$ then return $\lightning$
    if $fresh = \mathsf{false}$ then return $\lightning$
    $\mathcal{Z} \leftarrow i$
    return $K_i$
```

Fig. 12. The experiment $\mathsf{Exp}^{\mathrm{k}}_{\mathsf{EE,EU}}(\mathbb{A})$ used for the hybrid argument in Lemma 3.

Note that a single step from $k-1$ to k in $\mathsf{Exp}^{\mathrm{k}}_{\mathsf{EE,EU}}$ corresponds to $\mathsf{Exp}^{\mathrm{URRND}\text{-}b_{\mathrm{Up}},b_{\mathrm{RRND}}}_{\mathsf{EE}}$, hence

$$\Pr\left[\mathsf{Exp}^{\mathrm{k-1}}_{\mathsf{EE,EU}}(\mathbb{A}) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathrm{k}}_{\mathsf{EE,EU}}(\mathbb{A}) = 1\right] = \mathsf{Adv}^{\mathrm{URRND}}_{\mathsf{EE,EU}}(\mathbb{A}) \leq \mathsf{Adv}^{\mathrm{URRND}}_{\mathsf{EE,EU}}(\mathbb{B}^{\mathbb{A}})$$

and we obtain

$$\mathsf{Adv}^{\mathrm{CRND}}_{\mathsf{EE,EU}}(\mathbb{A}) \leq q_{\mathrm{Up}} \cdot \mathsf{Adv}^{\mathrm{URRND}}_{\mathsf{EE,EU}}(\mathbb{B}^{\mathbb{A}}).$$

□

A.3 Introducing Multiple Key Sequences (CRND$_\ell$ Security)

We extend the experiment $\mathsf{Exp}^{\mathrm{CRND}\text{-}b}_{\mathsf{EE}}$ by allowing multiple sequences of keys: in addition to the key index j, we introduce also the sequence index h. Each sequence represents a router in an onion circuit, and we denote the proxy with index $h = 0$. The quantities i_h denote how many queries have been made for router h to the oracle Up. The resulting experiment $\mathsf{Exp}^{\mathrm{CRND}_\ell\text{-}b}_{\mathsf{EE}}$ corresponds to Fig. 5 and, by a standard hybrid argument, we obtain Lemma 4.

Lemma 4 (CRND **security** $\Longrightarrow$ CRND$_\ell$ **security).** *Let* $(\mathsf{EE},\mathsf{EU})$ *be an updatable tweakable split-domain cipher. Then there exists a simple fully black box reductions* $\mathbb{B}$ *such that, for all* CRND$_\ell$ *adversaries* $\mathbb{A}$ *against* $(\mathsf{EE},\mathsf{EU})$*,*

$$\mathsf{Adv}^{\mathrm{CRND}_\ell}_{\mathsf{EE,EU}}(\mathbb{A}) \leq \ell \cdot \mathsf{Adv}^{\mathrm{CRND}}_{\mathsf{EE,EU}}(\mathbb{B}^{\mathbb{A}}).$$

A.4 UIV+'s UPDT Security in Terms of PRF Security of F

The last missing piece is bounding the security of the update mechanism (Definition 3) based on the PRF Security of F.

Lemma 5 (PRF **Security of** $\mathsf{F} \implies$ UPDT **security).** *Consider* $\mathsf{UIV+}$ *as updatable tweakable split-domain cipher. Then there exists a simple fully black box reductions* $\mathbb{B}$ *such that, for all* UPDT *adversaries* $\mathbb{A}$ *against* $\mathsf{UIV+}$,

$$\mathsf{Adv}^{\mathrm{UPDT}}_{\mathsf{UIV+}}(\mathbb{A}) \leq \mathsf{Adv}^{\mathrm{PRF}}_{\mathsf{F}}(\mathbb{B}^{\mathbb{A}}).$$

Proof (sketch). Recall that, by definition,

$$\mathsf{Adv}^{\mathrm{UPDT}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = \Pr\Big[\mathsf{Exp}^{\mathrm{URRND\text{-}0,1}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big] - \Pr\Big[\mathsf{Exp}^{\mathrm{URRND\text{-}1,1}}_{\mathsf{EE},\mathsf{EU}}(\mathbb{A}) = 1\Big] \,,$$

so $\mathbb{A}$ always has direct access to $\mathsf{ED}^H_K()$ and $\mathsf{EE}^H_K()$, whereas access to $\mathsf{EU}_K()$ might be replaced by randomness (depending on the "update" challenge bit). All three algorithms share the same key, but in the case of $\mathsf{UIV+}$ the underlying tweakable blockcipher and tweakable pseudorandom function $\mathsf{F}_S()$ keys are independent of each other, so reduction $\mathbb{B}$ can simply sample the key of the tweakable blockcipher on its own. The reduction then uses the real $\mathsf{F}_S()$-oracle to answer $\mathsf{ED}^H_K()$ and $\mathsf{EE}^H_K()$ queries (always with tweak 0) and the challenge $\mathsf{F}_S()$-oracle to answer $\mathsf{EU}_K()$ queries (always with tweak 1). The different tweaks force domain separation, ensuring that there will be no overlap between the real and challenge oracle queries, proving the statement. $\square$