Proceedings on Privacy Enhancing Technologies ; 2020 (2):336–357

Sebastian Lauer*, Kai Gellert*, Robert Merget, Tobias Handirk, and Jörg Schwenk

T0RTT: Non-Interactive Immediate Forward-Secret Single-Pass Circuit Construction

Abstract: Maintaining privacy on the Internet with the presence of powerful adversaries such as nation-state attackers is a challenging topic, and the *Tor* project is currently the most important tool to protect against this threat. The *circuit construction protocol* (CCP) negotiates cryptographic keys for Tor circuits, which overlay TCP/IP by routing Tor cells over n onion routers. The current circuit construction protocol provides strong security guarantees such as *forward secrecy* by exchanging $\mathcal{O}(n^2)$ messages.
For several years it has been an open question if the same strong security guarantees could be achieved with less message overhead, which is desirable because of the inherent latency in overlay networks. Several publications described CCPs which require only $\mathcal{O}(n)$ message exchanges, but significantly reduce the security of the resulting Tor circuit. It was even conjectured that it is impossible to achieve both message complexity $\mathcal{O}(n)$ and forward secrecy immediately after circuit construction (so-called immediate forward secrecy).
Inspired by the latest advancements in zero round-trip time key exchange (0-RTT), we present a new CCP protocol *Tor 0-RTT (T0RTT)*. Using modern cryptographic primitives such as *puncturable encryption* allow to achieve immediate forward secrecy using only $\mathcal{O}(n)$ messages. We implemented these new primitives to give a first indication of possible problems and how to overcome them in order to build practical CCPs with $\mathcal{O}(n)$ messages and immediate forward secrecy in the future.

Keywords: Tor, onion routing, circuit construction, nTor handshake, 0-RTT, puncturable encryption

DOI 10.2478/popets-2020-0030
Received 2019-08-31; revised 2019-12-15; accepted 2019-12-16.

***Corresponding Author: Sebastian Lauer:** Ruhr University Bochum, Bochum, Germany, E-mail: sebastian.lauer@rub.de
***Corresponding Author: Kai Gellert:** Bergische Universität Wuppertal, Wuppertal, Germany, E-mail: kai.gellert@uni-wuppertal.de
Robert Merget, Jörg Schwenk: Ruhr University Bochum, Bochum, Germany, E-mail: {robert.merget, joerg.schwenk}@rub.de
Tobias Handirk: Paderborn University, Paderborn, Germany, E-mail: thandirk@mail.uni-paderborn.de

1 Introduction

Providing the ability to browse the web anonymously is important to guarantee freedom-of-speech and freedom-of-information in authoritarian states, and to protect civil rights activists all over the world. Maintaining this ability has become more and more challenging in the last years, with more and more nation-state adversaries establishing Internet surveillance programs to monitor their citizens.

Onion Circuit Construction. To protect against such surveillance, overlay networks based on the concept of *onion routing* can be used. In onion routing networks, the *onion proxy* (OP) wraps each message with multiple layers of encryption. Then, the message is transported through a path of nodes, called *onion routers* (OR), in an overlay network where each node decrypts one layer and sends the decrypted message to the next node until it reaches its destination. To encrypt a message in such a way, the onion proxy must negotiate cryptographic keys with each of the onion routers on the selected overlay path. Establishing those keys is the task of the *(onion) circuit construction protocol* (CCP), an $n+1$ party protocol performed between the onion proxy and the n onion routers.

In early versions of onion routing protocols, RSA encryption [46] was used in the circuit construction. This allowed for *single-pass circuit construction* by simply encrypting the symmetric key to be used with the current OR, and the cell to be forwarded to the next OR, with the public RSA key of the current OR. Single-pass circuit construction has message complexity $\mathcal{O}(n)$ and is therefore very efficient, but came at a large cost in security: If the secret key of an onion router was compromised, all data relayed to that OR could be decrypted.

Eventual vs. Immediate Forward Secrecy. To ensure that protected information of *closed* sessions or circuits stays secure even if an attacker compromises the long-term secret of an onion router, modern protocols aim to provide the property of *forward secrecy*

(FS) [31]. In 2007, the concept of FS in onion routing protocols was introduced and it distinguishes between two cases, *eventual forward secrecy* and *immediate forward secrecy* [41]. If a circuit construction protocol achieves FS *a certain time period after the circuit has been closed* by replacing the long-term keys of a router, then the protocol achieves *eventual FS*. If FS is achieved *immediately after the circuit is closed*, then the protocol provides *immediate FS*.

Currently the nTor protocol [26] is used for circuit construction, where each of the three nTor instances used in one circuit construction implements a variant of an authenticated Diffie–Hellman key exchange. This CCP achieves immediate FS, but has message complexity $\mathcal{O}(n^2)$.

In this paper, we apply novel results in zero round-trip time key exchange (0-RTT) to construct a single-pass circuit construction protocol with immediate forward secrecy. 0-RTT protocols allow a sender to transmit cryptographically protected data within the first message, without prior exchange of key establishment messages [33]. We show that our approach offers many advantages compared to other single-pass circuit constructions. Additionally, we investigate the feasibility of our construction against the currently most important onion routing network, the *Tor* protocol [22].

Achieving Immediate FS with Puncturable Encryption. For our construction, we use a novel cryptographic primitive called *puncturable encryption* [17, 18, 29, 32]. This primitive is an advanced variant of public key encryption (PKE) schemes comparable to RSA [46]. Like RSA, a *puncturable encryption scheme* has a key pair (pk, sk) with a public key pk and a secret key sk. The public key is static and publicly known. The secret key, however, alters with every decryption. If sk was used to decrypt ciphertext c^*, it is subjected to a transformation Punct which transforms it into a new secret key sk': $sk' \leftarrow \mathsf{Punct}(sk, c^*)$. This modified secret key can be used to decrypt *all* ciphertexts c created with the help of pk, except for ciphertext c^*. Repeatedly "puncturing" the secret key allows to stepwise revoke decryption capabilities of the secret key. This primitive was introduced in [29], and can be implemented using bilinear pairings.

In our construction, we use an efficient variant called *puncturable key encapsulation mechanism* [17, 18]. A *key encapsulation mechanism* (KEM) is a generalization of the concept of PKE, inspired by the ElGamal scheme [24]. In an ElGamal scheme, the sender of a message does not choose a symmetric key to encrypt it with the public key of the receiver, but the symmetric key is the result of a computation on a randomly chosen value and the sender's public key. By using this generalization in our construction, we keep it generic and open for future performance improvements in puncturable KEMs.

Typically, immediate FS in key exchange protocols is achieved by performing an ephemeral Diffie–Hellman Key Exchange (DHKE) [20], e.g. in TLS-DHE, IPSec IKE, or in the Tor nTor-Protocol. However, this comes at the cost of a two-message handshake *before* the first message can be encrypted.

Intuitively, immediate FS can be achieved with puncturable encryption (or puncturable KEMs) if the Punct operation is performed on the secret key immediately after the decryption of c, but before the completion of the handshake. If this is the case, then an adversary will only learn sk', and with this modified secret key it will not be able to decrypt the intercepted ciphertext c, due to the properties of puncturable encryption.

Our Contribution. Single-pass circuit construction has a message complexity of $\mathcal{O}(n)$, whereas all currently deployed immediate FS circuit construction schemes have a message complexity of $\mathcal{O}(n^2)$ (cf. Figure 1 (a)). Due to significant delay and latency inherent to message transmission on complex overlay networks such as Tor, it is desirable to reduce this message complexity.

The general assumption in prior work on single-pass circuit construction was that it is *impossible* to achieve immediate FS in a single-pass circuit construction [13, 14, 37]. This is indeed true if the onion routers' secret keys are *static*. Boyd and Gellert [10] give a more comprehensive discussion on forward secrecy in the context of single-pass protocols. In this work, we apply the results work of Günther *et al.* [32] to circuit construction protocols and disprove this assumption by proposing a new non-interactive single-pass circuit construction protocol that utilizes *non-static* secret keys and achieves *immediate FS* based on puncturable KEMs (cf. Figure 1 (b)). Our results can be summarized as follows:

- We present a *generic construction* for a single-pass circuit construction protocol, T0RTT, which can be instantiated with any IND-CCA-secure puncturable KEM. This protocol is the proposal of a multi-hop 0-RTT protocol.
- Our generic construction is the *first* to achieve immediate forward secrecy *immediately* after the circuit construction if an onion router updates its secret key non-interactively.
- We provide a *refined version* of the security model of Kate *et al.* [36], which includes definitions from

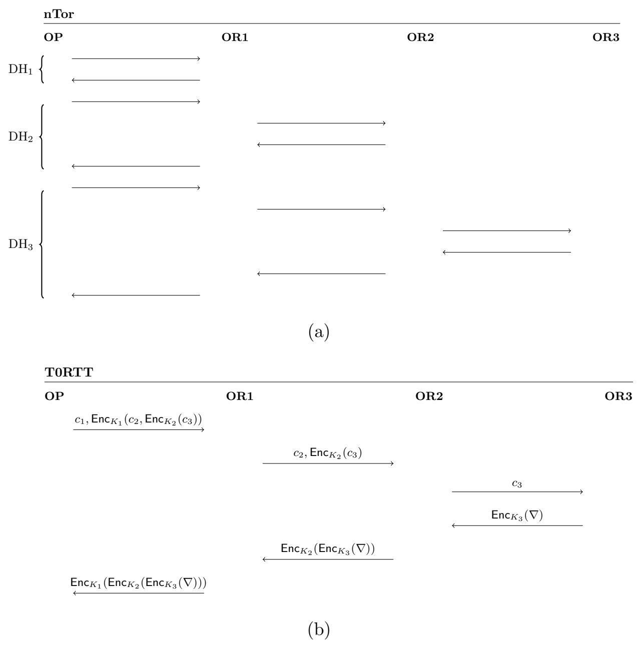
nTor
OP
OR1
OR2
OR3
DH_1
DH_2
DH_3
(a)
T0RTT
OP
OR1
OR2
OR3
$c_1, \mathsf{Enc}_{K_1}(c_2, \mathsf{Enc}_{K_2}(c_3))$
$c_2, \mathsf{Enc}_{K_2}(c_3)$
c_3
$\mathsf{Enc}_{K_3}(\nabla)$
$\mathsf{Enc}_{K_2}(\mathsf{Enc}_{K_3}(\nabla))$
$\mathsf{Enc}_{K_1}(\mathsf{Enc}_{K_2}(\mathsf{Enc}_{K_3}(\nabla)))$
(b)

Fig. 1. Simplified comparison of message patterns in nTor (a) and an abstraction of T0RTT (b), a single-pass circuit construction protocol. In order to achieve FS in (a) we have to perform three ephemeral DHKE, the second and third exchange being relayed over one and two onion proxies, respectively. In T0RTT, c_i contains all necessary information to exchange a forward-secret symmetric key.

Camenisch and Lysyanskaya [11]. To simulate large-scale adversaries we consider a network in which all but one routers are corrupted by an attacker. Additionally, the peculiarities of immediate forward secrecy are captured by the security model.

– We prove the security of T0RTT even if a *non-negligible* amount of errors occur during the circuit construction that leads to a termination of the circuit by introducing the novel concept of *dummy onions*.
– To examine the feasibility of our constructions in the Tor network, we implemented the latest puncturable KEM presented in the work of Derler *et al.* in [17, 18] to derive results on the computational overhead introduced through puncturable KEMs. We thus compare the performance of our constructions with the current Tor CCP implementation, and with other single-pass circuit constructions from the literature [13, 14, 36, 37].

Related Work on Puncturable Encryption and 0-RTT Protocols. Since puncturable encryption has been formally introduced by Green and Miers [29], several new construction have been proposed [17–19, 32]. So far, puncturable encryption has shaped many results in the area of 0-RTT protocols, where encrypted payload data is send non-interactively along with key material for key establishment.

One of the most prominent results is the work by Günther *et al.* [32]. They showed that non-interactive forward secrecy in 0-RTT protocols can be achieved using puncturable KEMs. This idea has since then been generalized to achieve forward secrecy in non-interactive protocols, such as email communication [49].

Aviram *et al.* [4] have developed a technique to overcome the lack of forward secrecy in protocols where session partners already share a symmetric secret, such as session resumption protocols (e.g., TLS 1.3 in the 0-RTT mode), while utilizing a symmetric variant of puncturable encryption called puncturable pseudorandom functions. Since they require preshared secrets between sender and recipients, their techniques are not applicable to our work.

2 Onion Routing

The preservation of privacy for users in the presence of powerful adversaries is a challenging topic, especially when the well-being of users might be at stake. To achieve this goal, *Tor* [22] was created. Tor is a circuit-based low-latency protocol for anonymous communication which tries to protect its users' privacy from non-global adversaries. In the Tor network, packets are routed through multiple servers which apply multiple layers of encryption to the actual data. This interleaved encrypted data is called an *onion*, whereas the whole process is called *onion routing*.

Despite many other constructions [25, 45], the most famous onion routing scheme is Tor. The foundations of the Tor network were laid by the work of Goldschlag, Reed, and Syverson in which they describe the basics and analyze the security of onion routing [27, 28, 44, 47]. The idea of onion routing was based on the ideas of anonymous communication presented by David Chaum in 1981 [15].

A route usually consists of three servers and is called a *circuit*. Servers within a circuit are called *onion routers* (OR), while the last server is oftentimes also called exit router. ORs are connected to each other via TLS and relay traffic through the network. Each router can be identified by an identity key and additionally, each router holds a short-term onion key. All information that is necessary to connect to an OR is stored in a descriptor file on a publicly available directory server.

A user can connect to the network by running an *onion proxy* (OP). An OP builds circuits in the network by choosing a path of nodes and exchanging symmetric keys with each node on the path. Application data is then sent over the path in so-called *cells* which are wrapped in multiple layers of encryption.

A command embedded within each cell instructs the receiver how to handle the contents of the cell. If an OP wants to send data to a server outside of the Tor network, it creates a *relay cell* which contains the application data encrypted with the symmetric keys that were exchanged during the circuit construction. Each OR then decrypts the payload of the relay cell with its key and forwards it accordingly.

Once the cell reaches the exit router the payload is forwarded to the target server. The answer from the target server is then encrypted and forwarded by each OR accordingly until it reaches the OP. The OP then decrypts the received cell with each of the respective symmetric keys. Along the path, each node only knows its predecessor and successor and for this reason, the user stays anonymous to all other nodes except the first node on the path. In this work, we focus on the process of circuit construction which is explained in more detail in the following section.

2.1 Single-Pass Circuit Construction

The first work on onion routing in 1996 [28] describes the circuit construction as follows. The client sends one onion message to the first node on the path in which each layer contains a symmetric key and the identifier for the next node on the path. All layers of this message are encrypted using the public keys of the respective nodes. The method to construct a circuit with only one message sent by the client, which is then passed through a path of nodes is called single-pass circuit construction. The biggest advantage of this construction is that the required number of messages to build a circuit is only $\mathcal{O}(n)$.

Unfortunately, the original circuit construction does not provide FS. If an attacker learns all secret keys of the nodes on the path, then the attacker is able to decrypt past communications as long as the secret key is not replaced by a new key. The current circuit construction handshakes used in Tor (TAP and nTor) provide immediate FS, but require $\mathcal{O}(n^2)$ messages to build a circuit of length n. To reduce this overhead of messages required for the circuit construction many works have been published in the last years.

Kate *et al.* propose a handshake in an identity-based infrastructure setting [36, 37] which achieves single-pass circuit construction and FS. Their paring-based onion routing (PB-OR) protocol uses identity-based encryption schemes, whereby the router's identity can be used as the public key. The main disadvantage of their proposal is the requirement of a trusted third party (TTP). The Key Generation Center (KGC), used in their setting, provides secret keys for the onion routers which are replaced after a certain time period. Corrupting the KGC means that an attacker corrupts the whole network. The authors present different solutions for this, but interaction between router and KGC is still needed to replace the current secret key after a certain time period. Hence, their protocol only achieves eventual FS.

In [14] the authors published a single-pass circuit construction scheme based on certificateless encryption. Their certificateless onion routing (CL-OR) uses a mix between public key encryption and identity-based encryption to achieve FS. In the certificateless setting the routers additionally hold a secret and public key pair which does not need to be certified. The property of eventual forward secrecy is achieved after a router replaces this pair of keys, which leads to interaction with the users.

In 2011, Catalano *et al.* presented a fully non-interactive single-pass circuit construction protocol with eventual forward secrecy based on *forward-secret identity-based encryption* [13]. Their single-pass circuit construction scheme provides efficient performance combined with static public keys. As in the works before, they only achieve eventual forward secrecy.

In the following section, we present the first non-interactive single-pass circuit construction with immediate FS without using a trusted third party.

Other Approaches. Besides the above mentioned single-pass circuit construction protocols there exist other approaches to improve the efficiency of the circuit construction in onion networks. More efficient protocols based on the TAP handshake combined with half-certified DHKE were presented in [41]. Unfortunately, in 2012 Goldberg *et al.* presented a work in which they show that a man-in-the-middle attack is possible against the new protocols [26]. Based on the ideas of Øverlier and Syverson, Goldberg *et al.* presented a new protocol which is known as the nTor protocol. In their work they have additionally shown that nTor is a secure one-way authenticated key exchange protocol. In the same year Backes *et al.* proposed ACE, an efficient protocol which reduces the computational cost of the nTor protocol at the expense of communication cost [5]. Just like nTor, the proposed protocol uses a telescoping approach for the circuit construction. Using the efficient message format *Sphinx*, Kate *et al.* are able to compress the messages needed in the single-pass constructions in [13, 36, 37]. However, their approach comes with higher computational costs for onion routers. Surveys on performance and security improvements for the Tor network are given in [2, 23].

3 Forward-Secret Single-Pass Circuit Construction

In this section we introduce the notion of *forward-secret single-pass circuit construction* (FSSPCC) and the security model for FSSPCC which we will use in this work to prove the security of our generic construction. For this we assume a network in which different ORs are located. Each OR is associated with a unique identifier ID_{OR} and has a secret and public key. Each onion proxy that wants to build a circuit in this network has access to the respective public keys of the ORs.

Optimal Circuit Length. Many academic results on circuit construction have considered circuits of length n [13, 14, 36, 37], while in practice only a length of $n = 3$ is deployed. This is mainly due to efficiency reasons and the fact that it is not known whether a circuit length of $n > 3$ does indeed increase security [43]. Even if greater path lengths would be allowed, it might pose a threat against anonymity:

– The path length could act as identifier if not all users commit to same length [7], and
– Mounting Denial-of-Security attacks gets easier [9].

In the following we will hence consider a path length of $n = 3$ to improve readability. Should any of our results also trivially hold for path lengths $n > 3$, we will explicitly state it.

Definition 1. A *forward-secret single-pass circuit construction protocol* FSSPCC with $n = 3$ hops consists of the following algorithms:

$\mathsf{FSSPCC.KGen}(1^\lambda) \to (pk, sk)$**.** On input of a security parameter λ, this algorithm outputs a key pair (pk, sk).

$\mathsf{FSSPCC.RunOP}(pk_1, pk_2, pk_3) \to (K_1, K_2, K_3, O_0)$**.** On input of $n = 3$ public keys, this algorithm outputs $n = 3$ session keys K_1, K_2, K_3, and an onion

$$O_0 = \ell_1 = (ID_{OR_1}, c_1, O_1), \text{ where}$$
$$O_1 = \mathsf{Enc}_{K_1}(\ell_2 = (ID_{OR_2}, c_2, O_2)), \text{ where}$$
$$O_2 = \mathsf{Enc}_{K_2}(\ell_3 = (ID_{OR_3}, c_3, O_3)), \text{ where}$$
$$O_3 = \mathsf{Enc}_{K_3}(\nabla),$$

where ∇ denotes the end of the path. We call the 3-tuple (ID, c, O) a *layer*.

$\mathsf{FSSPCC.DecOR}(sk, c, O) \to (K, sk', \ell)$**.** On input of a secret key sk, a ciphertext c and an onion O the algorithm outputs a session key K, a (potentially modified) secret key sk', and a layer ℓ that is either a tuple consisting of an identifier ID, a ciphertext c' and an onion O', or equal to ∇, indicating the end of the routed path.

We call an FSSPCC *correct* if all onions are routed correctly (i.e., all identifiers are decrypted correctly and all onions reach their intended destination; cf. [11, Def. 3]) and all ciphertexts decrypt their original encrypted messages. Additionally, we allow for an non-negligible correctness error of ε.

An FSSPCC protocol between an onion proxy and three onion routers is used as follows. First, the ORs generate a key pair using $(pk, sk) \xleftarrow{\$} \mathsf{FSSPCC.KGen}(1^\lambda)$. The OP then chooses a path of three different onion routers and uses their public keys to run FSSPCC.RunOP to produce the session keys K_1, K_2, K_3 and an onion of the form $O_0 = (ID_{OR_1}, c_1, \mathsf{Enc}_{K_1}(ID_{OR_2}, c_2, \mathsf{Enc}_{K_2}(ID_{OR_3}, c_3, \mathsf{Enc}_{K_3}(\nabla)))$. On input of the onion O_0 the first router on the path runs FSSPCC.DecOR with input of its current secret key, the ciphertext c_1 and the onion O_1 and outputs the session key K_1 which is used to remove the first layer of encryption. Additionally, the OR computes a potentially altered secret key sk'. Then, the new onion is send to the next router on the path. Each router repeats this process until a router decrypts the symbol ∇. Then the last router OR_3 computes a confirmation message of the form $\mathsf{Enc}_{K_3}(\top)$, where $\top$ is a special confirmation symbol, which indicates the success of the circuit construction. This onion is send to the previous router on the path back to the OP and each OR adds a layer of encryption using the computed session keys.

See Figure 2 for a more detailed description of a FSSPCC protocol run with three nodes.

Security Model for FSSPCC. We have to consider two separate security goals: (1) The security of the key agreement between the onion proxy OP and each onion

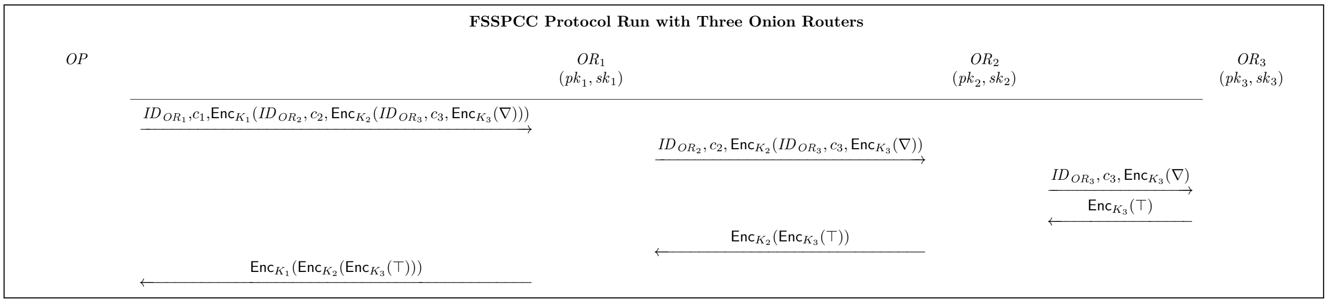
FSSPCC Protocol Run with Three Onion Routers
OP
OR_1 (pk_1, sk_1)
OR_2 (pk_2, sk_2)
OR_3 (pk_3, sk_3)
$ID_{OR_1}, c_1, \mathsf{Enc}_{K_1}(ID_{OR_2}, c_2, \mathsf{Enc}_{K_2}(ID_{OR_3}, c_3, \mathsf{Enc}_{K_3}(\nabla)))$
$ID_{OR_2}, c_2, \mathsf{Enc}_{K_2}(ID_{OR_3}, c_3, \mathsf{Enc}_{K_3}(\nabla))$
$ID_{OR_3}, c_3, \mathsf{Enc}_{K_3}(\nabla)$
$\mathsf{Enc}_{K_3}(\top)$
$\mathsf{Enc}_{K_2}(\mathsf{Enc}_{K_3}(\top))$
$\mathsf{Enc}_{K_1}(\mathsf{Enc}_{K_2}(\mathsf{Enc}_{K_3}(\top)))$

Fig. 2. Messages sent during the circuit construction, where ∇ denotes the end of a path, and $\top$ is a special confirmation symbol.

router. We must especially show that in contrast to previous proposals for single-pass circuit construction, we achieve *immediate forward secrecy* here. (2) The security of the overall circuit construction protocol should not be compromised, especially the *unlinkability* of circuits if at least one OR is not under the control of the adversary.

3.1 Adversarial Model

Our adversary is the classical cryptographic adversary who actively monitors all traffic on arbitrary sections of the Internet and who may delete, modify, insert or rearrange messages sent over this network. Our adversary is, however, not capable of performing traffic analysis, because it only controls a small part of the Internet at the same time. In addition, we provide it with the capability to corrupt and learn their secret keys.

3.2 Two-Party Security Goals: Key Secrecy and Immediate FS

A Tor circuit should remain indistinguishable from other Tor circuits even if an adversary controls two out of three onion routers. From this main security requirement (detailed in Section 3.3) two necessary cryptographic security requirements on the key establishment protocol performed between the onion proxy OP and the honest onion router OR can be derived:

- **Key Secrecy**: The key negotiated between an onion proxy OP and an onion router OR cannot be computed by any other party including the adversary, since otherwise, the Tor circuit would become traceable.
- **Immediate Forward Secrecy**: If an honest onion router OR is seized by the adversary and its secret key gets compromised, OR can be removed from the Tor directories to protect future circuits. However, the seizure should not endanger the anonymity of previously established and closed Tor circuits.[1]

Key Secrecy. Key secrecy (cf. [36]) can be defined as follows.

Definition 2. Let $\mathcal{A}$ be an active adversary who controls the network over which OP and OR communicate. To be precise, $\mathcal{A}$ can read, delete, modify or rearrange the sequence of all messages sent over this network. Let Π be a server-only authenticated key exchange protocol executed between client C and server S, and let pk_S be the public key of S used to authenticate S.

We say that Π guarantees *key secrecy* if the success probability of $\mathcal{A}$ in computing the session key K established between C and S is negligible.

Immediate Forward Secrecy. The current implementation of the nTor protocol achieves forward secrecy in a strong sense, which can be defined as follows.

Definition 3. Let $\mathcal{A}$ be an active adversary who controls the network over which OP and OR communicate. To be precise, the $\mathcal{A}$ can read, delete, modify or rearrange the sequence of all messages sent over this network. Let Π be a server-only authenticated key exchange protocol executed between client C and server S, and let pk_S be the public key of S used to authenticate S.

We say that Π provides *immediate forward secrecy* if, when $\mathcal{A}$ learns the secret key sk_S corresponding to

1 Note that this goal depends on the number of parties. If we consider immediate FS as a two-party security goal, we only need that corruption of an onion router does not affect key secrecy. If, however, we consider multiple parties, we also need to ensure that protected information such as anonymity still remains protected after corruption. This is implicitly reflected in the formalization of cryptographic unlinkability in Section 3.3.

the public key pk_S at time τ, then his success probability in computing any of the session keys K established between C and S before time τ does not increase.

For a more formal definition of FS, please see [34] and [35]. Please note that this definition does cover nTor, but does not cover the notion of eventual FS introduced in [41]. In our definition, the public key pk_S of S remains unchanged when $\mathcal{A}$ learns the secret key (as for nTor), whereas in eventual FS the public key must be changed at exactly the same time τ when the secret key is revealed.

3.3 N-Party Security Goals

Since we focus on onion routing and the Tor network in this work, any forward-secret single-pass circuit construction protocol should provide the following properties in presence of an adversary who "*can observe some fraction of network traffic; who can generate, modify, delete, or delay traffic; who can operate onion routers of his own; and who can compromise some fraction of the onion routers*" [22].

In this work we adapt the model used in [13, 36, 37] and refine it so that the model represents the Tor network as realistically as possible. In our security game on the property of unlinkability we therefore consider an honest onion router that maintains many incoming and outgoing connections.

We will later instantiate our generic protocol with a puncturable KEM that contains a non-negligible correctness error. Hence, we also have to consider this failure probability in our model. Our aim is to show that even if this error occurs, all circuits that have been established still provide unlinkability.

- **Integrity:** Integrity requires that even for an onion created by an adversary, the path length of this onion cannot exceed the maximum path length or the circuit construction fails (cf. [11, Def. 4]).
- **Cryptographic Unlinkability:** This property is given if no adversary can link messages between a sender and receiver if there exists at least one honest node on the circuit path. A formal definition of this property will be given below. As stated in [36], network-level linking attacks are excluded.

Unlinkability of Unclosed Circuits. Unlinkability ensures that an attacker is not able to link the OP and the final OR if there exists at least one honest node in the path. This property obviously needs to be ensured for all *closed* circuits. It is, however, not clear if a failed circuit construction might affect unlinkability. We are indeed able to relax this property for unclosed circuits. Namely, should the construction of a circuit fail, it does not immediately affect unlinkability. We argue that unlinkability for a closed connection has to be given among other (potentially failed) circuits. To be precise, unlinkability would only be at risk if an attacker is able to ensure the failing of all but one connection in the network. In this case, the attacker would trivially be able to break unlinkability of this one connection.

Circuit Position Hiding. In the literature [5, 13, 36, 37], there are other security goals mentioned such as *circuit position hiding*, which is achieved if it is not possible to learn the position of a router in the circuit by observing the incoming traffic of the router.

Intuition of Cryptographic Unlinkability. To provide anonymity, onion networks should ensure that there is no link between a sender of a circuit construction message and the last router on the path. Since timing attacks or traffic analysis have a high probability to find a link between sender and receiver, we only consider *cryptographic unlinkability* as explained in [13, 36, 37]. For this, consider a network in which multiple users send onion messages over a path with three ORs and an adversary controls all but one honest OR. Before we provide a formal definition for the property of unlinkability, we would like to give a brief intuition for our security experiment. In the security game played between an adversary and a challenger, the adversary gains full control over the routers in the onion network except for one router which is controlled by the challenger. This represents an honest node in an onion network. The adversary can actively build arbitrary circuits even over the honest node by interacting with the challenger. In the second phase of our experiment, the attacker is given two circuits in which the honest node acts as the middle node in the path and the decryption of one of those onions (to be specific, it is given the onion decrypted by the honest node). In contrast to the security experiment used in [13, 36, 37], we give the adversary access to the honest node's modified secret key immediately after the challenger decrypts the challenge onion for the honest node. Note that the adversary now controls the *entire* network. This simulates a corruption of the honest router after creating a circuit and thus models immediate FS according to Definition 3.

Unlinkability Security Experiment $UL^{unlink}_{\mathsf{FSSPCC}}(\mathcal{A})$.

Setup: The challenger $\mathcal{C}$ generates the key pair $(pk_H, sk_H) \xleftarrow{\$} \mathsf{FSSPCC.KGen}(1^\lambda)$ of the honest router OR_H and sends the public key pk_H to the attacker. The attacker then sends a list of public keys $pk_1, \ldots, pk_n$ to the challenger.

Phase 1: In this phase the attacker may build circuits using the secret and public keys it received in the setup phase. For decapsulations and decryption of onions destined for the honest node, the attacker may interact with the challenger by sending onions (OR_H, c, O) and receives the layer ℓ from the output of $\mathsf{FSSPCC.DecOR}(sk_H, c, O)$.

Challenge: The attacker may then start Phase 2 by sending an indicator symbol $\top$ to the challenger. Upon receipt of this symbol, the challenger computes two onions sent over the honest router of form

$$O_0^j = \ell_1^j = (ID_{OR_v}, c_1^j, O_1^j), \text{ where}$$
$$O_1^j = \mathsf{Enc}_{K_1}(\ell_2^j = (ID_{OR_H}, c_2^j, O_2^j)), \text{ where}$$
$$O_2^j = \mathsf{Enc}_{K_2}(\ell_3^j = (ID_{OR_w}, c_3^j, O_3^j)), \text{ where}$$
$$O_3^j = \mathsf{Enc}_{K_3}(\nabla),$$

with $j \in \{0,1\}$, $(v, w) \xleftarrow{\$} [n] \times [n]$ and $v \neq w$ by invoking $\mathsf{FSSPCC.RunOP}(pk_v, pk_H, pk_w)$. Note that the challenger computes all values (c_i^j, O_i^j) and thus knows them even without knowing sk_v, sk_w. Both ℓ_1^0 and ℓ_1^1 are send to the attacker.

The challenger now flips a coin $b \xleftarrow{\$} \{0,1\}$, decrypts both onions $(K_2, sk', \ell_3^j) = \mathsf{FSSPCC.DecOR}(sk, O_2^j)$.

– If $b = 0$: The attacker receives ℓ_3^0 along with the secret key sk' of the honest router OR_H.

– If $b = 1$: The attacker receives ℓ_3^1 along with the secret key sk' of the honest router OR_H.

Note that the attacker is now in possession of the honest router's secret key and thus controls the entire network.

Guess: $\mathcal{A}$ may output a guess b'.

We define the advantage of an adversary $\mathcal{A}$ to win the cryptographic unlinkablity game $UL^{unlink}_{\mathsf{FSSPCC}}(\mathcal{A})$ as

$$\mathtt{Adv}^{unlink}_{\mathsf{FSSPCC},\mathcal{A}}(\lambda) := \left|\Pr[b = b'] - \frac{1}{2}\right|.$$

Definition 4. We say the circuit construction protocol Π provides *cryptographic unlinkability* if the advantage for any PPT attacker $\mathcal{A}$ to win experiment $UL^{unlink}_{\mathsf{FSSPCC}}(\mathcal{A})$ is negligible.

Limitation of the Model. In our model, the client always chooses a new path through the network should it establish a new circuit, that is, an adversary should not be able to recognize that a client attempts to build another circuit. Mounting attacks that utilize tricking onion routers to fail connection establishment with honest clients hence requires some sort of traffic analysis by the adversary. Similar to previous research (e.g., [13, 36, 37]), we assume that our adversary is not able to perform traffic analysis. Hence, we only achieve cryptographic unlinkability rather than "real-world unlikability".

This is a general modeling problem and not limited to our work. Our current modeling techniques are limited to ideal-world constraints that do not properly model and capture attack vectors as mentioned above.

4 T0RTT

In this section we construct a secure FSSPCC protocol, which provides immediate FS. To this end, we introduce Bloom filter KEMs, a variant of puncturable KEMs, and symmetric encryption as building blocks. After that, we present a generic construction based on the aforementioned building blocks.

4.1 Building Blocks

Bloom Filter KEMs. For efficiency reasons, we consider a special variant of puncturable KEMs: *Bloom filter key encapsulation mechansims* (BFKEM) [17, 18]. BFKEMs achieve highly efficient puncturing procedures at the cost of a non-negligible correctness error when decapsulating.

Definition 5. A *Bloom filter key encapsulation mechanism* BFKEM consists of four probabilistic polynomial-time algorithms $\mathsf{BFKEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{Punct})$ with the following properties.

$\mathsf{KGen}(1^\lambda, m, k) \to (pk, sk)$**.** On input of a security parameter λ, parameters m and k, the algorithm outputs a key pair (pk, sk).

$\mathsf{Encap}(pk) \to (K, c)$**.** On input of a public key pk, the algorithm outputs a symmetric key $K \in \{0,1\}^*$ and the encapsulated symmetric key $c \in \{0,1\}^*$.

$\mathsf{Decap}(sk, c) \to K$**.** On input of a secret key sk and the encapsulated symmetric key c, the algorithm outputs a symmetric key $K \in \{0,1\}^* \cup \{\bot\}$.

$\mathsf{Punct}(sk, c) \to sk'$. On input of a secret key sk and the encapsulated symmetric key c, the algorithm outputs a (potentially modified) secret key sk'.

In this paper we use the BFKEM construction introduced in [17] to instantiate our generic circuit construction protocol (we refer to Appendix D for a complete description). The construction presented there allows for non-negligible correctness errors. We will show that these errors can be tolerated and do not impact security.

Definition 6. We say that a BFKEM is *correct with an ε-error* if for all $\lambda, m, k \in \mathbb{N}$, for all $(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda, m, k)$, and for all $(K, c) \xleftarrow{\$} \mathsf{Encap}(pk)$, we have that $\mathsf{Decap}(sk, c) = K$. Moreover, for any sequence $i = 0, \ldots, n-1$ (where n is determined by m, k) of invocations of $sk' \leftarrow \mathsf{Punct}(sk, c_i)$ for any $c_i \neq c$ it holds that $\Pr[\mathsf{Decap}(sk', c) = \bot] \leq \varepsilon(m, k)$ where ε is some (possibly non-negligible) bound.

It is easy to see, that all known BFKEM constructions [17, 18] are correct with an ε-error equal to the false-positive probability of the Bloom filter, as the computation of c is independent of the sequence c_i. Security is defined via the IND-CCA security game for BFKEM [18] described in Appendix B.

Additionally, we need a property called *publicly-checkable puncturing*, also defined by Derler *et al.* in [18], which enables checking if a ciphertext can be decapsulated *without* access to the secret key.

Definition 7. Let $\mathcal{Q} = (c_1, \ldots, c_w)$ be any sequence of ciphertexts. We say that BFKEM allows *publicly-checkable puncturing*, if there exists an efficient algorithm $\mathsf{CheckPunct}$ with the following correctness property:

1. Run $(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda, m, k)$.
2. Compute $C_i \xleftarrow{\$} \mathsf{Encap}(pk)$ and $sk := \mathsf{Punct}(sk, c_i)$ for all $i \in [w]$.
3. Let c be any ciphertext. We require that

$$\bot = \mathsf{Decap}(sk, c) \iff \bot = \mathsf{CheckPunct}(pk, \mathcal{Q}, c).$$

Derler *et al.* show that all currently known constructions of BFKEMs have this property. Note also that any puncutrable KEM with perfect correctness achieves this property trivially by checking whether $c \in \mathcal{Q}$.

Symmetric Encryption. Another building block used in our circuit construction protocol to ensure the security of an onion routing system is symmetric encryption. A *symmetric encryption scheme* consists of three algorithms $\mathsf{ENC} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. Due to space constraints, we provide the standard definitions of symmetric encryption in Appendix A.

4.2 T0RTT

In this section we describe our generic protocol. We stress that our protocol may be instantiated with *any* puncturable KEM. However, in order to capture the technical challenges of instantiating our protocol with a puncutrable KEM that features a non-negligible correctness error, we briefly discuss these challenges before presenting our protocol.

Unlinkability in the Presence of a Correctness Error. Consider our unlinkability experiment presented in Section 3.3, where the challenger constructs two challenge circuits and the attacker wins if it can link the circuits. The attacker is able to observe both incoming and outgoing messages of the honest onion router. Assume now that one of the two challenge circuits can be processed correctly, while the other fails during decryption at the honest node.

If the attacker is able to recognize that processing of an onion has failed (e.g., by outputting an error symbol or aborting the protocol) *and* be able to retroactively check whether one of the incoming messages for the honest router is *destined to fail* during decryption, it could trivially link the failed connection (and thus the succeeded connection as well). The technical challenge of achieving unlinkability in the presence of a non-negligible correctness error is hence to strip the attacker of the capability to recognize such behavior.

Unfortunately, all currently known constructions of BFKEMs allow to check in advance whether an incoming message is destined to fail during decryption as the randomness of a ciphertext is sent in the clear.[2] This recognizing property could by avoided by, for example, encrypting the randomness with a standard public key encryption scheme (e.g., RSA encryption). This approach would, however, cost us forward secrecy. As soon as an attacker comes into possession of the honest router's secret key, it can again retroactively check if an incoming ciphertext will fail. This could (in theory)

2 More technical: An attacker can keep track of the randomness r of every ciphertext sent to the honest router this enables the attacker to compute a copy of the honest router's Bloom filter which in turn allows it to predict if a ciphertext can be decapsulated.

be overcome by using another instance of a puncturable encryption scheme dedicated to the ciphertext randomness, but would render the scheme even more inefficient, both in storage and running time. This approach is in our opinion infeasible.

We propose to hide that a failure has happened. Instead of, for example outputting a failure symbol or aborting the protocol, the honest router could output a "dummy onion," an onion that looks correct to the adversary, but was fabricated by the router. This technique is related to the so-called "cover traffic" that originates from a slightly different context. It is used in anonymous communication to blend the sender's traffic into the noise of random cover traffic (e.g., [30]), thus hiding when it is really sending. We will utilize the "dummy onion" to hide that a decapsulation has failed.

Dummy Onions. On a technical level, dummy onions will look as follows. Let O be a received onion that will fail during processing. In this case, the onion router generates a dummy onion by first selecting a random router. The dummy onion contains a newly encapsulated key for the randomly chosen router and the encrypted message ∇, indicating the end of the circuit path. This concept ensures 1) that the router's output has the correct format and is therefore indistinguishable from an onion which was correctly decapsulated 2) that even if the dummy onion will fail at a later position in the path, the circuit construction protocol will terminate eventually.

We can easily bound the probability of termination. Let ε be the worst-case failure probability. We can bound the termination probability by $1-\varepsilon^M$, where M is the number of consecutive failures. It is easy to see, that the termination probability is overwhelmingly high for real-world parameters such as $\varepsilon = 1/1000$. The additional network load induced by dummy onions is thus negligible in comparison to the regular network load.

Note that computing a dummy onion is not more expensive than properly decapsulating a ciphertext when instantiated with a BFKEM as recommended in Section 6.1. All known BFKEMs only use symmetric computations (i.e., evaluation of universal hash functions and table look-ups) to recognize a decapsulation failure. As encapsulation and decapsulation are similarly expensive, the router does not suffer additional computational load when computing the "dummy onion" in comparison to a proper decapsulation.

We would like to point out that the use of a dummy onion is not possible if at the same time the property of "circuit position hiding" is required, since each router has to output an onion with the correct format according to its position in the circuit. The property of "circuit position hiding" means that in an onion routing network it should not be possible for a router to estimate the approximate position in a circuit. However, preventing routers from learning their position is impossible in a single pass circuit construction protocol with $n = 3$ and can be explained as follows. In a 3-hop circuit, nodes may check whether the sender of a message is a non-router. With this knowledge, a router can determine if it acts as the first node. The last router on the path decrypts the symbol ∇ which trivially leads to the conclusion that its the last router on the path. If a router acts neither as an entry nor as an exit node, then its position in a 3-hop circuit can logically only be that of the middle node. For this reason we are convinced that the instantiation with dummy onions is justified for 3-hop circuits, but we would like to point out that this solution is unsuitable for longer circuits.

Our Construction. In the following we describe the protocol for our BFKEM-based circuit construction for $n = 3$ onion routers. The protocol can be instantiated with any IND-CCA-secure BFKEM scheme. Advantages and disadvantages of different BFKEM constructions will be discussed in Section 6.1.

Definition 8. Let $\mathsf{BFKEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{Punct})$ be a Bloom filter key encapsulation mechanism and $\mathsf{ENC} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ a symmetric encryption scheme. We construct a forward-secret single-pass circuit construction protocol FSSPCC as follows:

$\mathsf{FSSPCC.KGen}(1^\lambda) \xrightarrow{\$} (pk, sk)$**.** Given a security parameter λ compute a key pair by running $(pk, sk) \leftarrow \mathsf{BFKEM.KGen}(1^\lambda, m, k)$, where the global parameters m and k denote the size of the Bloom filter and the number of universal hash functions, respectively.

$\mathsf{FSSPCC.RunOP}(pk_1, pk_2, pk_3) \rightarrow (K_1, K_2, K_3, O_0)$**.** On input of three public keys, the OP computes the following:

1. For each $i \in [3]$, the OP computes $(K_i, c_i) \leftarrow \mathsf{BFKEM.Encap}(pk_i)$.
2. After computing the symmetric keys the client builds the onions $(O_0, \ldots, O_3)$ as

$$O_0 = \ell_1 = (ID_{OR_1}, c_1, O_1), \text{ where}$$
$$O_1 = \mathsf{Enc}_{K_1}(\ell_2 = (ID_{OR_2}, c_2, O_2)), \text{ where}$$
$$O_2 = \mathsf{Enc}_{K_2}(\ell_3 = (ID_{OR_3}, c_3, O_3)), \text{ where}$$
$$O_3 = \mathsf{Enc}_{K_3}(\nabla),$$

$\mathsf{FSSPCC.DecOR}(sk, c, O) \to (K, sk', \ell)$. On input of the current secret key of the router OR, a ciphertext c and an onion O, the router computes the output of $\mathsf{BFKEM.Decap}(sk, c)$.

- In case of a decapsulation error, the router outputs a dummy onion according to its position as follows:
  - If OR is the first router on the path. OR randomly chooses two distinct identities $ID_{OR'_2}$ and $ID_{OR'_3}$ with pk'_2 and pk'_3, computes $(K'_2, c'_2) \leftarrow \mathsf{BFKEM.Encap}(pk'_2)$ and $(K'_3, c'_3) \leftarrow \mathsf{BFKEM.Encap}(pk'_3)$. We define the layer as $\ell = (ID_{OR'_2}, c'_2, \mathsf{Enc}_{K'_2}(ID_{OR'_3}, c'_3, \mathsf{Enc}_{K'_3}(\nabla))$ and output $(\perp, \ell, sk)$.
  - Else OR randomly chooses one identity $ID_{OR'_3}$ which has to be different from the identity of its predecessor. Then OR computes $(K'_3, c'_3) \leftarrow \mathsf{BFKEM.Encap}(pk'_3)$ and defines $\ell = (ID_{OR'_3}, c'_3, \mathsf{Enc}_{K'_3}(\nabla))$. Output is $(\perp, \ell, sk)$.
- Otherwise, the router punctures its secret key $sk' \leftarrow \mathsf{BFKEM.Punct}(sk, c)$ to achieve immediate FS. Using the derived symmetric key, the OR decrypts the current layer of the onion message and forwards the message to the next router on the path.

In an FSSPCC protocol, a router recognizes that it is the last router on the path after decrypting ∇. In case of a decapsulation error, a router would not be able to decrypt this message and therefore does not retrieve the information that it has to send the first message back to the onion proxy. In our construction the last router outputs an onion with the encrypted ∇ symbol. This approach does not affect the cryptographically unlinkability property, since the receiver of the regular message would be the known predecessor.

Remark on Instantiation. We want to stress once more that our construction is generic and can be instantiated with any puncturable KEM. Only puncturable KEMs with non-negligible correctness error [17, 18] need to utilize the dummy onion technique to prevent failure detection. All other known puncturable KEMs [19, 29, 32] provide perfect correctness and thus even allow for an extension of the path length to $n > 3$.

5 Security Analysis

In this section we discuss the security of our construction. Since our construction trivially achieves key secrecy and immediate FS, the main focus is to prove cryptographic unlinkability.

5.1 Two-Party Security

Key Secrecy. The key secrecy property follows from the properties of the used puncturable KEM.

Theorem 1. If one of the IND-CCA-secure puncturable KEMs from [17–19, 32] is used for key establishment, then key secrecy is guaranteed.

Proof. (Sketch) The puncturable KEM mentioned in the theorem provide IND-CCA security, i.e., the key that is established is indistinguishable from a random value, and can thus not be computed from the intercepted messages. This even holds for CCA adversaries which may invoke the Decap operation several times, thus the result holds in our weaker adversarial model. See [17–19, 32] for detailed proofs. □

Immediate FS. Immediate FS follows from the fact that Encap is a probabilistic algorithm, and that the Punct operation prohibits decapsulating a ciphertext twice.

Theorem 2. If a puncturable KEM with key secrecy is used for key establishment, and if the combination of Decap and Punct is implemented as one atomic operation, then immediate FS is guaranteed.

Proof. (Sketch) Our first assumption, which is satisfied by all KEM and puncturable KEM schemes proposed in the literature, is that the probabilistic Encap algorithms have entropy, say δ bit, to produce real pseudorandom output. Thus the probability $2^{-\delta}$ that the same pair (K, c) is computed twice during the lifetime of the key pair (pk, sk) can be made negligible, by choosing δ appropriately. This large entropy is also necessary to avoid decapsulation failures by the punctured secret key.

For each encapsulated key c that the adversary may record while observing the circuit construction protocol, the secret key sk is immediately punctured after decapsulation of c. We may now distinguish two cases: (1) The adversary has learned sk prior to an atomic call to $(\mathsf{Decap}(sk, c), \mathsf{Punct}(sk, c))$. That case does *not* violate the definition of immediate FS (cf. Def. 3), because the adversary learned sk at time τ, but the key K was established *after time* τ. (2) The adversary has learned sk' after the atomic call to $(\mathsf{Decap}(sk, c), \mathsf{Punct}(sk, c))$. In this case, the new secret key sk' cannot be used to decapsulate K anymore; a call to $\mathsf{Decap}(sk', c)$ will return $\perp$. Since the puncturable KEM provides key secrecy, there is no other way for the adversary to compute K. This completes the proof. □

5.2 N-Party Security

Circuit Position Hiding. We distinguish two cases depending on the instantiation. Should the instantiation have a correctness error (and thus use dummy onions), we need to sacrifice the property of circuit position hiding. Otherwise, routers would not be able to fabricate onions. Note that the current Tor circuit construction protocol also does not satisfy this goal. In [11] the authors show that an attacker is always able to derive information about the router's position by looking at the ciphertext's size.

Should the instantiation, however, be perfectly correct, we can achieve circuit position hiding by applying the techniques described in [11].

Cryptographic Unlinkability. The core security result of this paper is a proof that our construction satisfies *unlinkability* as the central cryptographic security goal of the circuit construction protocol for all circuits *established* prior to some point in time τ, even if

- the adversary controls two out of three onion routers, and
- the adversary learns the secret key of the third router at time τ.

In the following we will prove that our generic construction provides cryptographic unlinkability if the underlying BFKEM scheme and the underlying encryption scheme are IND-CCA secure. In our proof, we need IND-CCA security for both of our building blocks to correctly simulate decryption queries of an adversary for onions. Decryption queries simulate a realistic attacker who can send encrypted onions to routers in the network and wait for the decrypted messages. This was already discussed in [13].

Theorem 3. Let BFKEM be a BFKEM that is correct with an ε-error and has publicly-checkable puncturing, and let ENC be a symmetric encryption scheme. For any efficient polynomial-time adversary $\mathcal{A}$ in the experiment $UL^{unlink}_{\mathsf{FSSPCC}}(\mathcal{A})$ there exist efficient polynomial-time algorithms $\mathcal{B}_1, \mathcal{B}_2$ such that

$$\mathtt{Adv}^{unlink}_{\mathsf{FSSPCC},\mathcal{A}}(\lambda) \leq 2\cdot\left(\mathtt{Adv}^{\text{IND-CCA}}_{\mathcal{B}_1,\mathsf{BFKEM}}(\lambda) + \mathtt{Adv}^{\text{IND-CCA}}_{\mathcal{B}_2,\mathsf{ENC}}(\lambda)\right).$$

Due to space constraints, we provide the complete proof in Appendix C.

6 Performance Analysis

We will now show how our construction can be instantiated and how it performs compared to other known single-pass circuit constructions.

6.1 Instantiation

Our construction presented in Section 4.2 can be instantiated with any IND-CCA-secure puncturable KEM. Furthermore, we require two properties for practical instantiations.

- **Fast decryption and puncturing procedures.** Known puncturable KEMs typically invoke a heavy computational load when decrypting a ciphertext. Since onion routers have to perform multiple decryptions within a short time, a puncturable KEM with as little computational load as possible is desirable.
- **Short ciphertexts.** The current specification allows 509 bytes of payload per cell [21], limiting the maximum size of a ciphertext. Even though it is being discussed whether this payload size should be increased in the future [39], short ciphertexts are desired.
- **Reasonably short secret keys.** Efficient puncturable KEMs often incur large secret keys up to several gigabytes in size due to heavy precomputations upon key generation. While this is often unavoidable, it is desirable to choose a scheme that generates secret keys of manageable size.

In the past, several puncturable KEMs have been proposed [17, 18, 29, 32]. Some of those constructions are not suitable for our protocol due to their drawbacks. The schemes proposed by Green and Miers [29] suffer from a decapsulation time that grows linear in the number of punctures, rendering them unsuitable to use in high-traffic scenarios. Similarly, the scheme proposed by Günther *et al.* and Derler *et al.* are impractical since puncturing takes up to several seconds for reasonable deployment parameters [19, 32].

A promising attempt for practical puncturable encryption schemes in high-traffic scenarios are Bloom filter key encapsulation mechanisms [17, 18]. Bloom filter key encapsulation mechanisms is a variant of puncturable KEMs that achieves highly efficient puncturing procedures (deleting few parts of the secret key) while keeping the secret key reasonably small. This, however, comes at the cost of a non-negligible correctness error that might occur when puncturing too often.

In order to better understand the properties of BFKEM, we will give a brief intuition of its concept before discussing suitable choices to instantiate our protocol. The core idea of BFKEM is to precompute an array of secret keys, where puncturing consists of deleting a few entries of this secret key array. Naïve constructions (e.g., one secret key per ciphertext) suffer under exponentially large secret keys. Using Bloom filters as a probabilistic data structure helps to overcome this obstacle. Intuitively, a Bloom filter allows keeping track on which parts of the secret keys have already been punctured while maintaining a shorter secret key array compared to naïve approaches (we provide a proper definition of Bloom filters in Appendix D).

Simplified, a typical BFKEM works as follows: Imagine a cryptographic primitive that upon initialization computes a public key pk and a master secret key. The master secret key msk is used to issue additional secret keys sk_i for arbitrary identities i. Encapsulation takes as input the public key pk and a (subset of) identities. Anyone who possesses a secret key sk_i, where i was used in the encapsulation, is able to decapsulate.[3]

For BFKEM we use a primitive with the aforementioned properties. We generate a BFKEM key pair by computing a public key pk and a master secret key msk. Then we use msk to compute secret keys for all possible identities. We store the computed identity secret keys in an array and discard msk.

To encapsulate, a client draws some randomness $r \leftarrow \mathcal{R}$ from a randomness space $\mathcal{R}$. This randomness r implicitly defines a subset of random identities under which the client encapsulates a fresh session key (e.g., by encrypting the session key with respect to each of the identities in the subset). The encapsulated key is then sent to the server along with the chosen randomness r. The server decrypts by recomputing the subset of chosen identities with r and choosing *one* existing identity secret key from its secret key array to retrieve the session key.

The puncturing procedure uses the randomness r to delete *all* identity secret keys associated with the subset of identities implicitly defined by r. This ensures that encapsulated keys cannot be decapsulated when the same r is used twice. Since each invocation of the puncturing procedure leads to the deletion of multiple parts of the secret key, it may happen that for an "unpunctured" randomness r' no identity secret keys are leftover, rendering decryption impossible.

A Bloom filter is a way to parametrize such constructions. That is, given a maximum number of punctures throughout the key pair's lifetime n and a tolerated correctness error p (after n punctures), we can compute the optimal size $m = -n \ln(p)/(\ln(2))^2$ of the secret key array. BFKEM is highly parameterizable and suitable parameters have to be chosen according to application and requirements.

While the puncturing procedure of Bloom filter encryption is highly efficient, most BFKEM schemes have ciphertexts that grow linearly in the size of the identity subset (i.e., we encrypt the session key for each identity separately). However, Derler *et al.* also propose a BFKEM based on identity-based broadcast encryption (IBBE) [17], which allows to aggregate encapsulated keys for several identities, yielding *constant size* ciphertexts (when instantiated with the constant-size ciphertext IBBE by Delerablée [16]).

The IBBE-based BFKEM instantiated with the Delerablée IBBE yields constant size ciphertexts that consist of four elements and constant size identity secret keys that consist of a single group element. In conclusion, we consider this construction to be a suitable choice for our protocol. For completeness, we provide the technical details of our chosen BFKEM in Appendix D.

6.2 Efficiency and Comparison

In order to compare our proposed construction to previous single-pass circuit constructions, we evaluated the computational costs required by nTor [26], PB-OR [36, 37], CL-OR [14], and NI-OR [13] and visualized our results in Table 1.

To evaluate the performance of our *T0RTT* protocol we developed a single core implementation in the C programming language based on the RELIC [3] framework. To configure RELIC we used the presets for Barreto-

3 On a technical level, this primitive is called an identity-based KEM.

Security Level	Times	nTor	PB-OR	CL-OR	NI-OR	T0RTT
	User	0.30	1.41	0.54	2.01	8.03
78 bit	OR	0.10	0.71	0.16	2.09	5.52
	Circuit Construction	1200.60	603.54	601.02	608.28	624.59
	User	0.30	2.1	0.84	3.12	13.79
112 bit	OR	0.10	0.94	0.28	2.92	9.07
	Circuit Construction	1200.60	604.92	601.68	611.88	641.00
	User	0.30	4.53	1.95	6.93	30.54
128 bit	OR	0.10	2.03	0.62	6.37	19.52
	Circuit Construction	1200.60	610.62	603.81	626.04	689.10

Table 1. Comparison of computation and execution times (in ms) for 3-hop circuits with a latency of 100 ms and a security level of 78, 112 and 128 bit, respectively. All measurements except T0RTT are estimates. The parameters for T0RTT are $n = 1\,000\,000$ and $p = 10^{-3}$.

Naehrig (BN) curves [6] with an estimated security level of 78, 112 and 128 bit, respectively, which are provided by RELIC's authors. The parameter choice for the BN curves is in line with the analysis of Menezes *et al.* [40] who advise for conservative parameter choices due to recent advances in solving the discrete logarithm problem in $\mathbb{G}_T$ [38]. For the evaluation we chose the parameters $m = 14\,377\,588$ and $k = 10$ which corresponds to a total number of $n = 1\,000\,000$ punctures over the lifetime of the public key and a tolerated correctness error $p = 10^{-3}$. Our implementation makes use of precomputation wherever possible. That is, each OR precomputes certain exponentiations for all elements from $\mathbb{G}_1$ and $\mathbb{G}_2$ of its public key. This allows the OR to significantly speed up the decapsulation since all exponentiations in both of those groups are with a base from the public key. Users cannot benefit from precomputation since all exponentiations depend on the public key of the ORs. In our proposed *T0RTT* construction the entire encapsulation performed by the user can be computed offline. The whole decapsulation process can be highly parallelized which would significantly decrease the computation time needed by an OR.

We did not implement the other circuit construction schemes but estimated their performance by measuring the runtime of the individual cryptographic operations. We consider the symmetric encryption required by all circuit constructions as computationally free since they are negligibly small in comparison to public key operations. This methodology is analogous to the comparison in [13] and considers the same required operations for the different single-pass circuit constructions. We again chose the same parameters as for the evaluation of our protocol implementation, i.e. BN curves with an estimated security level of 78, 112, and 128 bit, respectively, using the RELIC library.[4] We evaluated the performance for the exponentiations in $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ and the pairings with the benchmarking code included in RELIC. We used OpenSSL[48] and X25519 for the performance estimations of nTor for all security levels, as OpenSSL's X25519 outperformed all implementations of smaller curves. All measurements were performed on a MacBook Pro running MacOS 10.14.6 with an Intel Core i5-6267U Processor with 2.9 GHz. We provide the measurements of the individual operations in Appendix E. A comparison of the properties of the different methods can be found in Table 2.

A ciphertext in our construction consists of one element from $\mathbb{G}_1$ and one from $\mathbb{G}_2$ together with two elements from $\{0,1\}^\lambda$. Depending on the security level the size of an element from $\mathbb{G}_1$ is 20, 32 and 48 bytes, respectively, and the size of an element from $\mathbb{G}_2$ is 40, 64 and 96 bytes, respectively. This results in a ciphertext sizes of 80, 124, and 176 bytes. In addition to the security level, the size of the public key also depends on the tolerated correctness error p. Again, assuming a correctness error $p = 10^{-3}$, the public key has a size of 700, 1120 and 1680 bytes, respectively. Given that there are less than 10 000 relays [42] the size of the public keys of all relays will be less than 20 MB.

While we are able to keep the public key and ciphertext relatively short, the array of secret keys is rather large. The relay has to precompute enough secret keys, such that at the end of the public key lifetime there are still enough secret keys left so that the failure probability is still smaller than the tolerated correctness error

4 Although the evaluated constructions are defined over Type-I bilinear pairings $\mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ one can transform them to using Type-III pairings $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ [1].

Feature	nTor	PB-OR	CL-OR	NI-OR	T0RTT
No. of messages	$n(n+1)$	$2n$	$2n$	$2n$	$2n$
Immediate PFS	✓	✗	✗	✗	✓
No TTP	✓	✗	(✓)	✗	✓
Static public key	✗	✓	✗	✓	✗

Table 2. Comparison of features for circuits of length n for the different circuit construction schemes.

p. That is, depending on the public keys lifetime and the number of requests n the relay has to serve within this lifetime, we are able to compute the optimal size of the secret key array, such that the chance of failure is p if the secret key has been punctured n times. The time needed to compute the secret key grows linearly in the size of the array. We measured the time it takes to compute a single entry of the array and thus we can estimate the time it takes to compute the whole array on a single core. We provide computations of the resulting secret key size and computation times in Table 3.

These computations show that our proposed T0RTT instantiation can perform considerably faster than the current nTor protocol, although computational intensive operations are used. The biggest drawback of our instantiation is the secret key size, which can grow to unmanageable sizes if the key is not frequently rotated on a busy OR. We believe that although the key can get quite large they could still be managed when the parameters for the ORs are carefully chosen. We remark that puncturable KEMs are a relatively novel area of research and hope to see further constructions that mitigate current drawbacks. More efficient puncturable KEMs immediately yield a more efficient FSSPCC protocol.

7 Issues and Solutions

One major drawback of our proposed T0RTT construction is the increased resource consumption in regards to CPU and RAM. This can be especially challenging for exit nodes and constrained relay-nodes. As shown by [42] a major part of the Tor network are very network constrained devices. With the assumption that a device with little bandwidth is likely also not strong in regards to computational power[5], large parts of the Tor network are very restricted in their processing power.

5 If this assumption does not hold, T0RTT would be even more efficient.

Typically, our construction has large secret keys upon initialization which decrease with each puncturing operation. The initial size of the secret key depends on the number of connections a node needs to serve in its lifetime. That is, the secret key for a high-traffic node is much larger than of a node with less traffic.

Additionally, our proposed construction has an increasing chance of failure with each puncturing operation, which would result in even higher circuit construction times. Recall that this chance of failure depends on how the key has been generated. That is, we can generate a key in such way, that *after* n puncturings the chance of decapsulation failure is $p = 1/1000$. A smaller value of p and larger values of n will both increase the secret key size, respectively.

For RAM-constrained devices, it might be infeasible to keep the initial secret key in its memory and it has to be outsourced to its hard drive, which in return decreases the performance of the proposed algorithm. Additionally, the large secret key is harder to manage than traditional Tor secret keys, since it is constantly changing, and a backup of a non-punctured key would break the immediate FS (while still achieving eventual FS after modifying or deleting the backup key as well). Should the server crash and thus lose its secret key, a new one must be generated. Note that the expensive computation of a new secret key can be avoided by precomputing and storing it on hard drive until needed.

Another issue concerns key propagation. Typically, key lifetimes overlap, so that clients have time to learn about new keys before the old one is discarded. If, however, a secret key is exhausted too quickly or lost due to a server crash, a new public key needs to be issued immediately for the node to still function. This requires both the server to adjust its public directory and the client to potentially download the new key material, sacrificing any performance gains at initial communication with the node.

We argue that these problems are most likely not a problem for the Tor protocol. Frequent key updates are already part of the established protocol. In the current default configuration, the Tor protocol updates the

Security Level	Lifetime of pk in days	Requests per day									
		10k		100k		1000k		10000k		100000k	
		m	t	m	t	m	t	m	t	m	t
78 bit	1	3	9 s	29	1.5 m	289	15.1 m	2886	2.5 h	28854	1 d
	7	21	1.1 m	202	10.6 m	2020	1.8 h	20198	17.6 h	201978	7.3 d
	14	41	2.1 m	404	21.1 m	4040	3.5 h	40396	1.5 d	403955	14.7 d
	28	81	4.2 m	808	42.3 m	8080	7 h	80791	2.9 d	807910	29.4 d
112 bit	1	5	15.6 s	47	2.6 m	462	26 m	4617	4.3 h	46167	1.8 d
	7	33	1.8 m	324	18.2 m	3232	3 h	32317	1.3 d	323164	12.6 d
	14	65	3.6 m	647	36.3 m	6464	6.1 h	64633	2.5 d	646328	25.2 d
	28	130	7.3 m	1293	1.2 h	12927	12.1 h	129266	5 d	1292655	50.5 d
128 bit	1	7	33.6 s	70	5.6 m	693	56 m	6925	9.3 h	69250	3.9 d
	7	49	3.9 m	485	39.3 m	4848	6.5 h	48475	2.7 d	484746	27.2 d
	14	97	7.9 m	970	1.3 h	9695	13.1 h	96950	5.5 d	969492	54.5 d
	28	194	15.7 m	1939	2.5 h	19390	1.1 d	193899	10.9 d	1938983	109.1 d

Table 3. Comparison of the secret key size $m = -n\ln(p)/(\ln(2))^2$ in MB and estimated time t needed for key generation for different lifetimes of the public key and number of requests within this lifetime $n \in \{10^4, 10^5, 10^6, 10^7, 10^8\}$ for a correctness error $p = 10^{-3}$ and a security level of 78, 112 and 128 bit, respectively.

public key already every 28 days. This number can be tweaked by relays to better fit their hardware constraints, meaning a smaller relay could update its key more frequently while a stronger relay could update less often.

Since the newly proposed scheme is a lot more computationally intensive, the new scheme must be hardened against Denial-of-Service attacks. A Tor relay should only process T0RTT messages while it still has resources left. If a relay is already busy it should fall back to the classic nTor handshake.

If clients wish to prevent falling back to nTor due to failing decryption they could theoretically send multiple onions at once. This would drastically reduce the chance for a failed decryption, at the cost of more encapsulations from the client and more network overhead. We discourage this construction since it increases the complexity of the protocol.

Another disadvantage, which should not go unmentioned, is the aspect of post-compromise security. After compromising an OR in the current setup of the Tor network, the adversary still has to interact actively as a Man-in-the-Middle attacker to read or alter messages. In our construction compromising an OR results in an adversary that can act passively to decrypt messages intended for the OR. However, replacing the secret key of the onion router once a month, does not only improve the performance but also strengthens the post-compromise security of T0RTT.

Possible Research Direction. We give a possible research direction in Appendix F.

8 Conclusion

In this paper, we disproved the assumption that it is impossible to achieve immediate forward secrecy for single-pass circuit construction. Despite the fact that current puncturable KEMs involve high computation costs and large secret keys, our performance evaluations have shown that the proposed protocol T0RTT is a plausible application scenario for this new cryptographic primitive.

Puncturable KEMs usually do not scale well, since they require a centralized architecture and get unmanageable with a growing number of connections. Neither of those drawbacks affect their use in onion networks. Especially in the Tor network, onion routers already replace their onion keys frequently and are usually computationally idle since they do not host other applications. Our evaluation showed that our proposed protocol is able to establish circuits faster than the current circuit construction protocol based on nTor handshake. In contrast to prior work, our protocol does not have to sacrifice security properties such as immediate forward secrecy to achieve this performance. Moreover, we do not require a trusted third party to be involved in the secret key generation.

Our proposed construction is only the first step in this direction. Additional improvements in the construction of puncturable KEMs can further improve our proposed generic approach. Although we propose this construction for the Tor protocol, other applications might benefit from similar approaches as well.

Acknowledgments. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2092 CASA – 390781972 and under project JA 2445/2-1. We thank our shepherd Wouter Lueks for many fruitful discussions, Tibor Jager for suggesting the use of dummy onions, and Fynn Dallmeier as well as all anonymous reviewers for their valuable comments.

References

[1] Masayuki Abe, Jens Groth, Miyako Ohkubo, and Takeya Tango. Converting cryptographic schemes from symmetric to asymmetric bilinear groups. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 241–260, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[2] Mashael Alsabah and Ian Goldberg. Performance and security improvements for tor: A survey. *ACM Comput. Surv.*, 49(2):32:1–32:36, September 2016.

[3] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[4] Nimrod Aviram, Kai Gellert, and Tibor Jager. Session resumption protocols and efficient forward security for TLS 1.3 0-RTT. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 117–150. Springer, 2019.

[5] Michael Backes, Aniket Kate, and Esfandiar Mohammadi. Ace: An efficient key-exchange protocol for onion routing. In *Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society*, WPES '12, pages 55–64, New York, NY, USA, 2012. ACM.

[6] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *SAC 2005: 12th Annual International Workshop on Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331, Kingston, Ontario, Canada, August 11–12, 2006. Springer, Heidelberg, Germany.

[7] Kevin Bauer, Joshua Juen, Nikita Borisov, Dirk Grunwald, Douglas Sicker, and Damon McCoy. On the optimal path length for Tor. In *HotPETS in conjunction with Tenth International Symposium on Privacy Enhancing Technologies (PETS 2010)*, Berlin, Germany, 2010.

[8] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[9] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of service or denial of security? In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 07: 14th Conference on Computer and Communications Security*, pages 92–102, Alexandria, Virginia, USA, October 28–31, 2007. ACM Press.

[10] Colin Boyd and Kai Gellert. A modern view on forward security. Cryptology ePrint Archive, Report 2019/1362, 2019. https://eprint.iacr.org/2019/1362.

[11] Jan Camenisch and Anna Lysyanskaya. A formal treatment of onion routing. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 169–187, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany.

[12] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 207–222, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

[13] Dario Catalano, Mario Di Raimondo, Dario Fiore, Rosario Gennaro, and Orazio Puglisi. Fully non-interactive onion routing with forward secrecy. *Int. J. Inf. Secur.*, 12(1):33–47, February 2013.

[14] Dario Catalano, Dario Fiore, and Rosario Gennaro. Certificateless onion routing. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *ACM CCS 09: 16th Conference on Computer and Communications Security*, pages 151–160, Chicago, Illinois, USA, November 9–13, 2009. ACM Press.

[15] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981.

[16] Cécile Delerablée. Identity-based broadcast encryption with constant size ciphertexts and private keys. In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 200–215, Kuching, Malaysia, December 2–6, 2007. Springer, Heidelberg, Germany.

[17] David Derler, Kai Gellert, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. Cryptology ePrint Archive, Report 2018/199, 2018. https://eprint.iacr.org/2018/199.

[18] David Derler, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 425–455, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[19] David Derler, Sebastian Ramacher, Daniel Slamanig, and Christoph Striecks. I want to forget: Fine-grained encryption with full forward secrecy in the distributed setting. Cryptology ePrint Archive, Report 2019/912, 2019. https://eprint.iacr.org/2019/912.

[20] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[21] Roger Dingledine and Nick Mathewson. Tor protocol specification.

[22] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[23] Roger Dingledine and Steven J. Murdoch. Performance Improvements on Tor or, Why Tor is slow and what we're going to do about it. https://www.torproject.org/press/presskit/2009-03-11-performance.pdf.

[24] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.

[25] Michael J. Freedman and Robert Morris. Tarzan: a peer-to-peer anonymizing network layer. In Vijayalakshmi Atluri, editor, *ACM CCS 02: 9th Conference on Computer and Communications Security*, pages 193–206, Washington D.C., USA, November 18–22, 2002. ACM Press.

[26] Ian Goldberg, Douglas Stebila, and Berkant Ustaoglu. Anonymity and one-way authentication in key exchange protocols. *Des. Codes Cryptography*, 67(2):245–269, May 2013.

[27] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Commun. ACM*, 42(2):39–41, February 1999.

[28] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In *Proceedings of the First International Workshop on Information Hiding*, pages 137–150, London, UK, UK, 1996. Springer-Verlag.

[29] Matthew D. Green and Ian Miers. Forward secure asynchronous messaging from puncturable encryption. In *2015 IEEE Symposium on Security and Privacy*, pages 305–320, San Jose, CA, USA, May 17–21, 2015. IEEE Computer Society Press.

[30] Tim Grube, Markus Thummerer, Jörg Daubert, and Max Mühlhäuser. Cover traffic: A trade of anonymity and efficiency. In *International Workshop on Security and Trust Management*, pages 213–223. Springer, 2017.

[31] Christoph G. Günther. An identity-based key-exchange protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology – EUROCRYPT'89*, volume 434 of *Lecture Notes in Computer Science*, pages 29–37, Houthalen, Belgium, April 10–13, 1990. Springer, Heidelberg, Germany.

[32] Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-RTT key exchange with full forward secrecy. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 519–548, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

[33] Britta Hale, Tibor Jager, Sebastian Lauer, and Jörg Schwenk. Simple security definitions for and constructions of 0-RTT key exchange. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17: 15th International Conference on Applied Cryptography and Network Security*, volume 10355 of *Lecture Notes in Computer Science*, pages 20–38, Kanazawa, Japan, July 10–12, 2017. Springer, Heidelberg, Germany.

[34] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[35] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. Authenticated confidential channel establishment and the security of TLS-DHE. *Journal of Cryptology*, 30(4):1276–1324, October 2017.

[36] Aniket Kate, Greg Zaverucha, and Ian Goldberg. Pairing-based onion routing. In *Proceedings of the 7th International Conference on Privacy Enhancing Technologies*, PET'07, pages 95–112, Berlin, Heidelberg, 2007. Springer-Verlag.

[37] Aniket Kate, Greg M. Zaverucha, and Ian Goldberg. Pairing-based onion routing with improved forward secrecy. *ACM Trans. Inf. Syst. Secur.*, 13(4):29:1–29:32, December 2010.

[38] Taechan Kim and Razvan Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 543–571, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[39] Nick Mathewson and Isis Lovecruft. Allow CREATE cells with >505 bytes of handshake data. https://dip.torproject.org/dgoulet/torspec/blob/master/proposals/249-large-create-cells.txt.

[40] Alfred Menezes, Palash Sarkar, and Shashank Singh. Challenges with assessing the impact of nfs advances on the security of pairing-based cryptography. In *International Conference on Cryptology in Malaysia*, pages 83–108. Springer, 2016.

[41] Lasse Overlier and Paul Syverson. Improving efficiency and simplicity of Tor circuit establishment and hidden services. In *Proceedings of the 7th International Conference on Privacy Enhancing Technologies*, PET'07, pages 134–152, Berlin, Heidelberg, 2007. Springer-Verlag.

[42] Tor Project. Tor Metrics. https://metrics.torproject.org/.

[43] Tor Project. Tor FAQ. https://www.torproject.org/docs/faq.html.en, November 2018.

[44] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE J.Sel. A. Commun.*, 16(4):482–494, September 2006.

[45] Marc Rennhard and Bernhard Plattner. Introducing morphmix: Peer-to-peer based anonymous internet usage with collusion detection. In *Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society*, WPES '02, pages 91–102, New York, NY, USA, 2002. ACM.

[46] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[47] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an analysis of onion routing security. In *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, pages 96–114, Berlin, Heidelberg, 2001. Springer-Verlag.

[48] The OpenSSL Project. OpenSSL: The open source toolkit for SSL/TLS, April 2003. www.openssl.org.

[49] Jianghong Wei, Xiaofeng Chen, Jianfeng Wang, Xuexian Hu, and Jianfeng Ma. Forward-secure puncturable identity-based encryption for securing cloud emails. In *European Symposium on Research in Computer Security*, pages 134–150. Springer, 2019.

A Building Block: Symmetric Encryption

Definition 9. A *symmetric encryption scheme* consists of three algorithms $\mathsf{ENC} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with the following properties.

$\mathsf{Gen}(1^\lambda) \to K$**.** On input of a security parameter λ, the algorithm outputs a key K.

$\mathsf{Enc}_K(m) \to C$**.** On input of a key K and a message m, the algorithm outputs ciphertext C.

$\mathsf{Dec}_K(C) \to m$**.** On input of a key K and ciphertext C, the algorithm recovers a message m or returns a failure symbol $\perp$.

We say that a symmetric encryption scheme is *correct* if for all m it holds that

$$\Pr[\mathsf{Dec}_K((\mathsf{Enc}_K(m)) = m \mid \mathsf{KGen}(1^\lambda) \to K] = 1.$$

Definition 10. Consider the following IND-CCA-security game played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$

1. The challenger generates a key $\mathsf{KGen}(1^\lambda) \to K$ and $\mathcal{A}$ is given access to the oracles $\mathsf{Enc}_K(\cdot)$ and $\mathsf{Dec}_K(\cdot)$. Using these oracles, $\mathcal{A}$ is able to encrypt (resp. decrypt) arbitrary messages (resp. ciphertexts).
2. $\mathcal{A}$ outputs two messages m_0 and m_1. The challenger flips a coin $b \xleftarrow{\$} \{0,1\}$ and encrypts $c \xleftarrow{\$} \mathsf{Enc}_K(m_b)$
3. The adversary receives c and may now use the oracles $\mathsf{Enc}_K(\cdot)$ and $\mathsf{Dec}_K(c')$ for $c' \neq c$.
4. Eventually, $\mathcal{A}$ outputs a guess b'

We define the advantage of an adversary to win the IND-CCA security game as

$$\mathtt{Adv}^{\text{IND-CCA}}_{\mathcal{A},\mathsf{ENC}}(\lambda) := \left|\Pr[b = b'] - \frac{1}{2}\right|.$$

B IND-CCA Security for BFKEM

Derler *et al.* describe the following IND-CCA security experiment for BFKEM [18].

Definition 11. The security of a BFKEM is defined by the following IND-CCA security game played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

1. Challenger $\mathcal{C}$ generates a fresh key pair $(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda, m, k)$. It computes $(c^*, K_0) \xleftarrow{\$} \mathsf{Encap}(pk)$ and selects $K_1 \xleftarrow{\$} \mathcal{K}$, where $\mathcal{K}$ is the symmetric key space. Additionally, it draws a random bit $b \xleftarrow{\$} \{0,1\}$ and sends (pk, c^*, K_b) to the adversary $\mathcal{A}$.
2. Adversary $\mathcal{A}$ may now ask a polynomial number of the following queries.
   - $\mathsf{Decap}(c)$: If $c = c^*$ the challenger outputs $\perp$. Otherwise, the challenger computes $K := \mathsf{Decap}(sk, c)$ and returns K to $\mathcal{A}$.
   - $\mathsf{Punct}(c)$: The challenger computes $sk' := \mathsf{Punct}(sk, c)$ and returns $\top$.
   - $\mathsf{Corrupt}$: The challenger aborts if $\mathsf{Punct}(c^*)$ has not been queried before. Otherwise, the challenger returns the current secret key sk to $\mathcal{A}$.
3. Eventually, $\mathcal{A}$ will output a guess b'.

We define the advantage of an adversary $\mathcal{A}$ to win the IND-CCA-security game as

$$\mathtt{Adv}^{\text{IND-CCA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda) := \left|\Pr[b = b'] - \frac{1}{2}\right|.$$

C Proof of Theorem 3

We will conduct this proof in a sequence of games between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. We start with the adversary playing the unlinkability security game. Over a sequence of hybrid arguments, we will stepwise transform the security game to a game where the challenger is independent of bit b. The claim then follows from bounding the probability of distinguishing any two consecutive games. Let $\mathtt{Adv}_i := |\Pr[b' = b] - 1/2|$ be the advantage of $\mathcal{A}$ in Game i.

Game 0. This is the original unlinkability security game $UL^{unlink}_{\mathsf{FSSPCC}}(\mathcal{A})$ and therefore it holds

$$\Pr[b = b'] = \frac{1}{2} + \mathtt{Adv}^{unlink}_{\mathsf{FSSPCC},\mathcal{A}} = \frac{1}{2} + \mathtt{Adv}_0.$$

Game 1. In this game, the challenger replaces the key used to encrypt the onion $(O_2^0 = \mathsf{Enc}_K(ID_{OR_w}, O_3^0, c_3^0))$ by a random value K^*. We claim that

$$\mathtt{Adv}_0 \leq \mathtt{Adv}^{\text{IND-CCA}}_{\mathcal{B}_1,\mathsf{BFKEM}}(\lambda) + \mathtt{Adv}_1.$$

Suppose an attacker $\mathcal{A}$ that can distinguish between Game 1 and Game 0. We use $\mathcal{A}$ to build an attacker $\mathcal{B}_1$ to break the IND-CCA security of the BFKEM scheme. $\mathcal{B}_1$ plays the IND-CCA security game and receives (pk, c^*, K_b). The attacker $\mathcal{B}$ then queries $\mathsf{Punct}(c^*)$ and $\mathsf{Corrupt}$ to receive the punctured secret key sk^*. The received secret key sk^* will be used as the secret key of the honest router OR_H. All other secret and public keys are generated by $\mathcal{A}$. Adversary $\mathcal{B}_1$ then interacts with $\mathcal{A}$ as described in the security experiment. Whenever the adversary $\mathcal{B}_1$ receives an tuple

(ID_{OR_H}, c, O), $\mathcal{B}_1$ runs $\mathsf{FSSPCC.DecOR}(sk_H = sk^*, c, O)$ and returns the decryption of the onion O or a dummy onion according to Definition 8. Additionally, the adversary $\mathcal{B}_1$ stores the received ciphertext c. After receiving the indicator symbol $\top$, $\mathcal{B}_1$ computes the challenge circuits but uses the key K_b to encrypt the onion $(O_2^0 = \mathsf{Enc}_{K_b}(ID_{OR_w}, O_3^0, c_3^0))$ with $c_2^0 = c^*$. Since $(ID_{OR_w}, O_3^0, c_3^0)$ is known to $\mathcal{B}_1$, the adversary can behave correctly in the challenge phase according to the security experiment. Here it should be mentioned that storing the ciphertexts c from Phase 1, allows the adversary $\mathcal{B}_1$ to use the publicly-checkable puncturing porperty of the BFKEM. Given the challenge ciphertext c^*, $\mathcal{B}_1$ can check whether O_2^0 can be decapsulated or adversary $\mathcal{B}_1$ must output a dummy onion where ∇ is encrypted under a random key.

$\mathcal{B}_1$ checks whether c^* can be decapsulated given the queries made by $\mathcal{A}$ so far. If it can, then it returns $\mathsf{Dec}(O_2^0)$ as above. Otherwise it samples a random identity $ID_{OR_3'} \overset{\$}{\leftarrow} \{ID_1, \ldots ID_n\} \setminus \{ID_{OR_v}\}$ and computes $(K_3', c_3') \overset{\$}{\leftarrow} \mathsf{BFKEM.Encap}(pk_3')$. Finally, the dummy layer $\ell = (ID_{OR_3'}, c_3', \mathsf{Enc}_{K_3'}(\nabla))$ is given to $\mathcal{A}$.

If K_b is the real key, then we are in Game 0 and if K_b is a random we simulate perfectly Game 1, and every attacker that can distinguish both games can be used to break the IND-CCA security of the BFKEM scheme.

Game 2. The next step in our proof is to replace the encryption key for onion $O_2^1 = \mathsf{Enc}_K(ID_{OR_{w'}}, O_3^1, c_3^1)$ by a random value K^*. We claim that

$$\mathtt{Adv}_1 \leq \mathtt{Adv}^{\text{IND-CCA}}_{\mathcal{B}_1,\mathsf{BFKEM}}(\lambda) + \mathtt{Adv}_2.$$

The proof of this conclusion works in a similar way to the proof given before.

Game 3. In Game 3 we replace the encrypted string in the onion $O_2^0 = \mathsf{Enc}_K(ID_{OR_w}, O_3^0, c_3^0))$ by a random string of the same length. Any adversary that can distinguish between Game 3 and Game 2 leads to an adversary that breaks the IND-CCA security of the encryption scheme. Therefore we have

$$\mathtt{Adv}_2 \leq \mathtt{Adv}^{\text{IND-CCA}}_{\mathcal{B}_2,\mathsf{ENC}}(\lambda) + \mathtt{Adv}_3.$$

Again, suppose an attacker $\mathcal{A}$ that can distinguish between Game 3 and Game 2. We will use this attacker to build an adversary $\mathcal{B}_2$ against the security of the symmetric encryption scheme. $\mathcal{B}_2$ plays against the IND-CCA challenger. $\mathcal{B}_2$ interacts with $\mathcal{A}$ as described in the experiment. After receiving the indicator symbol, $\mathcal{B}_2$ builds the challenge circuits but replaces the onion O_2^0 with the following. $\mathcal{B}_2$ picks a random string Γ with $|\Gamma| = |(ID_{OR_w}, O_3^0, c_3^0)|$ and sets $m_0 = (ID_{OR_w}, O_3^0, c_3^0)$ and $m_1 = \Gamma$. Then, $\mathcal{B}_2$ uses m_0 and m_1 to receive a challenge ciphertext c and sets $O_2^0 = c$. The rest of the experiment can be simulated by $\mathcal{B}_2$ correctly . If the received ciphertext is the encryption of $(ID_{OR_w}, O_3^0, c_3^0)$ we are in Game 2, else we simulate Game 3, and every attacker that can distinguish both games can be used to break the IND-CCA security of the encryption scheme.

Game 4. As before, we replace the encrypted string in the onion O_1^1 by a random string of the same length. Thus,

$$\mathtt{Adv}_3 \leq \mathtt{Adv}^{\text{IND-CCA}}_{\mathcal{B}_2,\mathsf{ENC}}(\lambda) + \mathtt{Adv}_4.$$

The proof of this claim is similar to the proof for Game 3.

We have now entirely replaced the plaintext and the keys of the encrypted onions by random values. The view of $\mathcal{A}$ in this game is independent from the chosen values and from b. Therefore,

$$\mathtt{Adv}_4 = 0.$$

Summing up all advantages above we can conclude

$$\mathtt{Adv}^{unlink}_{\mathsf{FSSPCC},\mathcal{A}}(\lambda) \leq 2\cdot\left(\mathtt{Adv}^{\text{IND-CCA}}_{\mathcal{B}_1,\mathsf{BFKEM}}(\lambda) + \mathtt{Adv}^{\text{IND-CCA}}_{\mathcal{B}_2,\mathsf{ENC}}(\lambda)\right).$$

□

Unlinkability and Dummy Onions. The unlinkability property states that the input of an onion cannot be linked to the output of an onion versa. To prove our construction secure we replace the input onions with random values in each game hop and show that an adversary that can detect these changes leads to an attacker that breaks the used primitive. In presence of a correctness error it might happen, that a dummy onion has to be generated by the reduction. If an adversary $\mathcal{A}$ is able to distinguish a real output from a random output, it would increase the advantage of $\mathcal{A}$ to distinguish the different games in our sequence-of-games which in turn would lead to an attacker that can break the security of the used BFKEM or encryption scheme. However, this would contradict the assumption that the respective primitive is secure.

D Instantiation in Detail

For completeness, we provide a brief overview of the algorithms used in our implementation. Most parts of this section are a shortened version of [18] (general definition of Bloom filters) and [17] (the construction). For more detailed explanations we recommend reading the corresponding sections of those publications.

In the following, we introduce Bloom filters as a building block. Based on this building block we describe the Bloom filter encryption scheme from identity-based broadcast encryption instantiated with Delerablée's identity-based broadcast encryption scheme [16].

Additionally, we use the modified Fujisaki–Okamoto transformation described in Section 2.6 of [18] to achieve IND-CCA security. Alternatively, the CHK-transformation [12] can be applied to the IBBE scheme to achieve a more efficient decryption with the drawback of a larger ciphertext.

Bloom Filters. A Bloom filter [8] is a probabilistic data structure that allows a compact representation T of a subset $\mathcal{S} \subseteq \mathcal{U}$. Formally, a Bloom filter can be constructed as follows.

Definition 12. A *Bloom filter* for set $\mathcal{U}$ consists of three probabilistic polynomial-time algorithms $\mathsf{BF} = (\mathsf{BFGen}, \mathsf{BFUpdate}, \mathsf{BFCheck})$ with the following properties.

$\mathsf{BFGen}(m,k) \to (H,T)$**.** On input two integers $m, k \in \mathbb{N}$, the algorithm samples k universal hash functions $H_1, \ldots, H_k$, where $H_j : \mathcal{U} \to [m]$ and defines $H := (H_j)_{j\in[k]}$ and $T = 0^m$. Output is (H,T).

$\mathsf{BFUpdate}(H,T,u) \to T'$**.** On input $H = (H_j)_{j\in[k]}$, $T \in \{0,1\}^m$ and $u \in \mathcal{U}$, the algorithm assigns $T' := T$. It updates the i-th bit of $T'[i] := 1$ for all $i = H_j(u)$ with $j \in [k]$. Output is T'.

$\mathsf{BFCheck}(H,T,u) \to b$**.** On input $H = (H_j)_{j\in[k]}$, $T \in \{0,1\}^m$ and $u \in \mathcal{U}$, the algorithm outputs a bit

$$b := \bigwedge_{j\in[k]} T[H_j(u)].$$

Relevant properties of Bloom filters include

- Perfect completeness: A previously added element will *always* be recognized by the Bloom filter,
- Compact representation: The size of the Bloom filter $|T| = m$ is independent of $\mathcal{S} \subseteq \mathcal{U}$. A larger set $\mathcal{S}$ increases only the false-positive probability,
- Bounded false-positive probability: The probability that a Bloom filter recognizes an element $u \notin \mathcal{S}$, is bounded by $(1 - e^{-k|\mathcal{S}|/m})^k$.

For a more precise treatment of Bloom filters in the context of Bloom filter encryption, we refer the reader to Section 2.1 of [18].

Construction. In [17], the authors have shown how to construct a BFKEM using any identity-based broadcast encryption scheme as a building block. We instantiate this construction with the identity-based broadcast encryption scheme by Delerablée [16] and apply the modified Fujisaki–Okamoto transformation of [18] (in combination with the transformation to a BFKEM with separable randomness) to achieve CCA-security.

Let $\mathsf{BF} = (\mathsf{BFGen}, \mathsf{BFUpdate}, \mathsf{BFCheck})$ be a Bloom filter. Let $(p, e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be public parameters of a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with prime order p and $|p| = \lambda$, where λ is the security parameter. Let $\mathcal{H} : \mathbb{Z}_p^* \to \mathbb{Z}_p^*$, $G : \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathcal{R} : \{0,1\}^* \to \mathbb{Z}_p^* \times \{0,1\}^{2\lambda}$ be cryptographic hash functions. We construct a Bloom filter key encapsulation mechanism $\mathsf{BFKEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{Punct})$ as follows:

- $\mathsf{KGen}(1^\lambda, m, k)$: The key generation algorithm generates a Bloom filter instance by running $(H,T) \leftarrow \mathsf{BFGen}(m,k)$. It chooses two generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ and a secret value $\gamma \leftarrow \mathbb{Z}_p^*$ and defines
$$pk_{\mathsf{IBBE}} := \left(w = g_1^\gamma, v = e(g_1, g_2), g_2, g_2^\gamma, \ldots, g_2^{\gamma^k}\right).$$
For each $i \in [m]$ it computes
$$sk_i = g_1^{\frac{1}{\gamma + \mathcal{H}(i)}}.$$
Finally, it discards γ and sets
$$pk := (H, pk_{\mathsf{IBBE}}) \text{ and } sk := \left(T, (sk_i)_{i\in[m]}\right).$$
- $\mathsf{Encap}(pk)$: Given a public key $pk = (H, pk_{\mathsf{IBBE}})$ with $pk_{\mathsf{IBBE}} = (w, v, g_2, g_2^\gamma, \ldots, g_2^{\gamma^k})$, it samples a random seed $S \xleftarrow{\$} \{0,1\}^\lambda$ and computes $(\rho, r, K) := \mathcal{R}(S)$. Next, it computes $K' := v^\rho$ and generates indices $i_j := H_j(r)$ for $(H_j)_{j\in[k]} := H$.
Finally, the algorithm computes a ciphertext $C' = (c_1, c_2, c_3)$ with
$$\begin{aligned} c_1 &:= w^{-\rho}, \\ c_2 &:= g_2^{\rho \cdot \prod_{j=1}^k (\gamma + \mathcal{H}(i_j))}, \\ c_3 &:= G(K') \oplus S \end{aligned}$$
and outputs (C, K), where ciphertext $C := (C', r)$.
- $\mathsf{Decap}(sk, c)$: The input is a secret key $sk = (T, (sk_i)_{i\in[m]})$ and ciphertext $C = (C', r)$. Again, it generates indices $i_j := H_j(r)$ for $(H_j)_{j\in[k]} := H$. If $\mathsf{BFCheck}(H,T,r) = 1$, then the algorithm returns $\perp$. Else, there exists at least one index $j^* \in [k]$ such that $sk_{i_{j^*}} \neq \perp$. The algorithm picks the smallest index j^* that meets the previous requirements and computes
$$K' = \left(e\left(c_1, g_2^{p_{j^*,r}(\gamma)}\right) \cdot e(sk_{i_{j^*}}, c_2)\right)^{\frac{1}{\prod_{j=1, j\neq j^*}^k \mathcal{H}(i_j)}},$$

where

$$p_{j^*,r}(\gamma) = \frac{1}{\gamma}\left(\prod_{j=1,j\neq j^*}^{k}(\gamma + \mathcal{H}(i_j)) - \prod_{j=1,j\neq j^*}^{k}\mathcal{H}(i_j)\right) \cdot$$

Next, it retrieves the seed $S = G(K') \oplus c_3$, and verifies that $(C, K) = \mathsf{Encap}(pk; S)$ (recomputing (C, K) with given S). If this does not hold, it outputs $\perp$. Otherwise, it outputs K.

– $\mathsf{Punct}(sk, c)$: Given a secret key $sk = (T, (sk_i)_{i\in[m]})$ and a ciphertext $C = (C', r)$, it invokes $T' = \mathsf{BFUpdate}(H, T, r)$ and defines

$$sk'_i := \begin{cases} sk_i, & \text{if } T'[i] = 0 \\ \perp, & \text{if } T'[i] = 1. \end{cases}$$

Finally, the algorithm returns $sk' = (T', (sk'_i)_{i\in[m]})$.

This construction is IND-CCA-secure if the (f, g, F)-GDDHE assumption, a variant of a generalization of a Diffie–Hellman exponent assumption, holds [16–18].

E Measurements

Table 4 provides measurements of the individual operations used for the analysis in Section 6.2. The runtime of the operations have been evaluated on a MacBook Pro running MacOS 10.14.6 with an Intel Core i5-6267U Processor with 2.9 GHz.

	78 bit	**112 bit**	**128 bit**
Exp. Tor [O][a]	0.05	0.05	0.05
Exp. in $\mathbb{G}_1$ [R]	0.08	0.14	0.31
Exp. in $\mathbb{G}_1$ [R,pc]	0.05	0.07	0.17
Exp. in $\mathbb{G}_2$ [R]	0.20	0.34	0.80
Exp. in $\mathbb{G}_2$ [R,pc]	0.12	0.22	0.47
Exp. in $\mathbb{G}_T$ [R]	0.42	0.63	1.34
Pairing [R]	0.71	0.94	2.03

Table 4. Time required by a single operation at different security levels in ms. R and O denotes that we measured the time using the RELIC [3] or OpenSSL [48], respectively. pc denotes the usage of precomputation.

a We used X25519 for all bit sizes, since OpenSSL's implementation of X25519 has better performance than any implementation of smaller curves we tested.

F Possible Research Direction

As an alternative way to solve the efficiency issues in Section 7, we propose a hybrid compatibility mode which is backward compatible with the previous nTor circuit construction. This mode would also allow a fluent integration into the current ecosystem. This backward compatibility mode would work as follows:

Each node in the Tor network submits its support for T0RTT to the public directory servers. A user can then choose his route through the Tor network as usual. Whenever it would now send an extend cell he additionally sends a *CreateT0RTT* cell which is encrypted with the appropriate keys and contains the data needed to further extend the circuit. If an OR is not willing to perform the T0RTT computations or is unable to since the BFKEM decapsulation failed, it can simply fall back to the nTor handshake.

This hybrid mode still allows constrained servers to perform the usual nTor construction while less constrained nodes can take advantage of the faster and less network intensive T0RTT circuit construction. Since the decryption failure is basically computationally free in the worst case our construction performs as bad as the current nTor handshake. Depending on the position of the nodes which do not speak T0RTT in the circuit the improvement is greater. However, we do *not* claim any security properties for this hybrid approach and leave this as future work.