Secure Opportunistic Multipath Key Exchange

Sergiu Costea
ETH Zürich
Zürich, Switzerland
sergiu-mihai.costea@inf.ethz.ch

Marios O. Choudary*
University Politehnica of Bucharest
Bucharest, Romania
marios.choudary@cs.pub.ro

Doru Gucea
University Politehnica of Bucharest
Bucharest, Romania
gucea.doru@gmail.com

Björn Tackmann
IBM Research – Zürich
Rüschlikon, Switzerland
bta@zurich.ibm.com

Costin Raiciu
University Politehnica of Bucharest
Bucharest, Romania
costin.raiciu@cs.pub.ro

ABSTRACT

The security of today's widely used communication security protocols is based on trust in Certificate Authorities (CAs). However, the real security of this approach is debatable, since certificate handling is tedious and many recent attacks have undermined the trust in CAs. On the other hand, opportunistic encryption protocols such as Tcpcrypt, which are currently gaining momentum as an alternative to no encryption, have similar security to using untrusted CAs or self-signed certificates: they only protect against passive attackers.

In this paper, we present a key exchange protocol, *Secure Multipath Key Exchange (SMKEX)*, that enables all the benefits of opportunistic encryption (no need for trusted third parties or pre-established secrets), as well as proven protection against some classes of active attackers. Furthermore, SMKEX can be easily extended to a trust-on-first-use setting and can be easily integrated with TLS, providing the highest security for opportunistic encryption to date while also increasing the security of standard TLS.

We show that SMKEX is made practical by the current availability of path diversity between different AS-es. We also show a method to create path diversity with encrypted tunnels without relying on the network topology. These allow SMKEX to provide protection against most adversaries for a majority of Alexa top 100 web sites.

We have implemented SMKEX using a modified Multipath TCP kernel implementation and a user library that overwrites part of the socket API, allowing unmodified applications to take advantage of the security provided by SMKEX.

CCS CONCEPTS

• **Security and privacy → Public key encryption**; **Security protocols**;

KEYWORDS

key exchange, opportunistic security, multi-path TCP

*I thank Christ our God for all His help during this work.

CCS'18, October 15–19, 2018, Toronto, ON, Canada

ACM ISBN 978-1-4503-5693-0/18/10…$15.00
https://doi.org/10.1145/3243734.3243791

ACM Reference Format:
Sergiu Costea, Marios O. Choudary, Doru Gucea, Björn Tackmann, and Costin Raiciu. 2018. Secure Opportunistic Multipath Key Exchange. In *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), October 15–19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3243734.3243791

1 INTRODUCTION

Secure communication is at the heart of our society today: we use secure e-mails, secure HTTP (HTTPS), secure shell (SSH), secure Voice-over-IP, secure messaging, etc. However, passive and active attackers still threaten the security of our communications. In particular, recent intelligence leaks and journalistic investigations have shown that the NSA controls some of AT&T's main routing facilities in order to analyze Internet traffic and long-distance calls [48, 51]. Rob Joyce from NSA has also admitted that NSA, as a persistent-threat adversary, will do everything it can to break into other people's machines and communications [25].

As a result, we have seen two major shifts in the landscape of communications: a) enabling HTTPS by default on most major websites and b) moving toward opportunistic encryption as the baseline instead of no encryption at all (plaintext). Unfortunately, the popular protocols that support these shifts (TLS and Tcpcrypt) still have their shortcomings.

TLS (Transport Layer Security) [13], as well as similar protocols such as QUIC [22], rely on third party servers – certificate authorities (CA) – to ensure the authenticity of public keys exchanged between two communicating parties (e.g., client and server), hence protecting against active man-in-the-middle (MITM) attacks. Yet, in the past years we have seen many issues with this system: a) CA's have been fooled to issue certificates to untrusted parties [33]; b) CA servers have been hacked [17, 20]; c) the mechanism for revoking bad or hijacked certificates is problematic [35]; d) CA's share their secret keys with other potentially insecure parties [7]. As a result, the protection that current TLS provides against active MITM attacks is debatable.

On the other hand, opportunistic encryption protocols such as Tcpcrypt [4] provide an improved security baseline for Internet communications [15]: rather than using no encryption, such protocols encrypt and authenticate data without requiring communicating parties to authenticate each others' public keys (i.e. no requirement for trusting third parties or pre-established secrets). This provides protection against passive attackers, but leaves the door open for

active ones (i.e. those that can tamper with the data). In essence, this is similar to using a compromised CA or self-signed certificates.

In this paper, we introduce Secure Multipath Key Exchange (SMKEX), a key exchange protocol that is an extension of the Diffie-Hellman key exchange to multiple channels. SMKEX provides opportunistic encryption with partial protection against active MITM attacks by leveraging the use of multiple *public* communication channels between communicating parties. Hence, our protocol has all the usability advantages of opportunistic encryption[1], while providing protection against several classes of active attackers. Furthermore, SMKEX can be easily extended to a trust-on-first-use (TOFU) setting and can be easily integrated with TLS, providing increased security for both TOFU and TLS-like protocols.

The security and practicality of our protocol is made possible by two main factors: *a*) the wide availability of path diversity between different AS-es [39]; *b*) the widespread adoption of Multipath TCP (MPTCP) [1] both on workstations and smartphones [37]. We have performed measurements of path diversity in several countries, which show that SMKEX can provide protection against local active attackers for a majority of Alexa top 100 websites. Furthermore, we can also use encrypted tunnels to create path diversity, allowing SMKEX to provide security even against nation-wide adversaries.

Overall, our work provides several useful contributions towards secure and practical opportunistic encryption: *a*) proof that SMKEX can resist active, but unsynchronized attackers, thus providing the highest security of an opportunistic encryption protocol to date (§6); *b*) evidence that existing path diversity in the Internet allows SMKEX to secure connections to many popular websites (§7); *c*) a method based on encrypted tunnels which allows SMKEX to protect against nation-wide adversaries (§7.2); *d*) an implementation of SMKEX using a modified MPTCP kernel implementation and a user library that overwrites the Socket API, allowing unmodified applications to use SMKEX (§8); *e*) description of integration with TLS and TOFU-based protocols (§10).

SMKEX builds on hash functions and is proven secure in the random-oracle model. We additionally describe in §D a variant of the protocol that is secure in the standard model.

2 ATTACKER MODEL

Generally, there are two types of network attackers: *passive attackers*, which only eavesdrop on a path, and *active attackers*, which additionally modify messages.

We consider the setting where a client connects to a server across multiple paths. If multiple attackers target the same path, we view them as a single attacker. If at least one of the attackers is active, we view all of them as a single active attacker. If only passive attackers target that path, we view them as a single passive attacker. In the Internet, attackers that are present on different paths can decide to collude. However, due to physical and national boundaries, attackers in different AS-es or countries might not be able to exchange data in real time. To capture such behavior, we define two types of relationships.

Definition 1 (Synchronized attackers). *Two attackers X_1 and X_2 are said to be synchronized (written X_1-X_2) if they can exchange messages between the start and end of a specific protocol session.*

[1]See the SoK paper of Unger et al. [49, Section III, Table I] for more details.

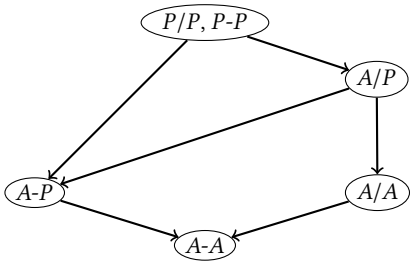

Figure 1: Map of all our possible adversaries based on intrusiveness (A, P) and communication capabilities $(-, /)$.

Definition 2 (Unsynchronized attackers). *Two attackers X_1 and X_2 are said to be unsynchronized (written X_1/X_2) if they can only exchange messages before the start and after the end of a specific protocol session.*

Unsynchronized attackers may agree over keys or data prior to protocol execution. They may also cooperate after termination of the session; this must not compromise the confidentiality of traffic.

2.1 Attack hierarchy for 2 paths

Based on attacker capabilities (active or passive) and the relationships between attackers (synchronized or unsynchronized) we classify attacks according to total attacker power. While the classification is simple for two paths (where we have only two attackers and one relationship between them), the complexity increases quickly as the number of paths grows. We discuss only the 2-path attacks in-depth, and in §C.2 introduce methods that reduce the complexity of any multipath attack to the 2-path case.

By looking at all possible combinations of attackers for the 2-path case, we obtain the following attack scenarios: P/P, P-P, A/P, A-P, A/A, and A-A.

An active attacker is stronger than a passive one, as it includes all the capabilities of the latter; additionally, synchronized attackers are superior to unsynchronized ones. We sort the 2-path attack scenarios based on attacker power in Figure 1. The arrows indicate the strength of the adversaries: an adversary at the end of an arrow is stronger (by the change of a single capability) than the one pointing to them. Furthermore, the arrows in reverse provide positive implications of security: for example, if we prove that a protocol provides some security property for the case A-P, this implies that the protocol also provides this property for the cases P-P, A/P and P/P. Note that P/P and P-P monitor all traffic without being able to inject messages during the execution of the protocol. However, because they can exchange information after the execution, they are equivalent. The A-P model requires that the active attacker cannot inject messages on the path controlled by the passive one, as otherwise the model collapses to A-A. In terms of real-world networks, A-P therefore requires an additional assumption such as the impossibility of address spoofing or strong timing guarantees.

We want to prove that a protocol provides a security property for some scenarios. We use the strict ordering of attacker capabilities to reduce the number of necessary proofs. Specifically, if we show our key exchange protocols are secure against A/A and A-P, it also implies they are secure against P/P, P-P, A/P. Opportunistic protocols

cannot achieve security against *A-A* attackers, so an opportunistic protocol secure against both *A/A* and *A-P* attackers achieves the best possible security according to our classification.

3 FUNDAMENTAL GOALS

In the following, we present the fundamental goals for secure opportunistic multipath key exchange in our model. In all our protocol analyses, we model the network as asynchronous, with no upper bound on the time required for a message to be delivered.

Secure key exchange. Our first goal (and the focus of our work) is to obtain a secure key exchange between two parties. We define a secure key exchange as a distinguishability game, adapting the model of Canetti and Krawczyk [6]. In particular, as they target authenticated key exchange and assume an adversary that has full control over the network, we describe restricted classes of adversaries that model *A/A* and *A-P* attackers, respectively. The guarantee formalized by our model is that any protocol session in which the adversary behaved as an *A/A* or an *A-P* attacker (and not as an *A-A* attacker) produces a key that is indistinguishable from a purely random one. This is the common security requirement for key-exchange protocols.

Forward and backward secrecy. Key exchange protocols are often required to also provide *forward secrecy* and *backward secrecy* (aka. future secrecy [11, 49]). Forward secrecy guarantees that an attacker that is able to compromise all the key material of communicating parties at a given time (e.g. temporary access to a device) is not able to decrypt data from previous communications [49, Section IV]. Often, forward secrecy is defined in terms of long-term secrets: compromising the long-term secrets should not allow an adversary to decrypt previously encrypted data. However, given the lack of long-term secrets in the basic version of our key exchange protocol, we define forward secrecy more generally, as Unger et al. [49, Section IV], meaning that an adversary that can compromise all the keying material cannot decrypt previously encrypted data. This definition also captures protocols using long-term keys (see our TOFU extension in §10).

Similarly, backward (or future) secrecy guarantees that an attacker that is able to compromise all the key material of communicating parties at a given time is not able to decrypt data from future communications [49, Section IV]. Again, this generic definition does not necessarily imply the use of long-term secrets, but only that an adversary be given all key material available at a given time.

4 UNDESIRABLE EXTENSIONS FOR MULTIPATH KEY EXCHANGE

Diffie and Hellman proposed the first efficient solution to exchange a secret key over a public path back in 1976 [14]. This protocol works as follows. Let Alice (A) and Bob (B) be the two parties that want to communicate (e.g. a client and a server) and let $\mathbb{G}$ be a cyclic group of prime order q, with generator g (with $|q|$, the bit length of q, large). Alice and Bob may agree on parameters $\mathbb{G}$, q, g beforehand. A chooses a secret exponent a in $\mathbb{Z}_q$ and sends the public value g^a (an element of $\mathbb{G}$) to B. Similarly, B chooses a secret exponent b in $\mathbb{Z}_q$ and sends g^b to A. They can now both obtain the secret key $K = g^{ab}$, by exponentiating the public value received from the other party to their secret exponent[2].

While the DH protocol is still the predominant key exchange mechanism used today, it is completely vulnerable to an active MITM attacker (M) that can intercept and modify the key exchange in order to set up different known keys with each of the participants. A common solution to prevent such attack, as used in TLS, is to sign the public value sent by the server and verify the signature through a chain of certificates [13]. But relying on certificates signed by third parties might lead to false security. Hence, we would like to establish secure communications without relying on trusted third parties or pre-established secrets (the context of opportunistic encryption). As we show in the next sections, this is possible by taking advantage of multiple communication paths.

In the following, we explain why previously proposed solutions for multipath key exchange, as well as some ideas that may seem good at first glance, fail to provide a secure key exchange against *A/A* or *A-P* adversaries.

Replicated Diffie-Hellman. A first possible approach would be to replicate the DH key exchange across all paths and then rely on majority voting to decide on the good key. But this would only work when there is a majority of passive attackers, which we cannot know in advance. Another option is to stop if we get different keys across some paths. However, in this case *A/A* adversaries could agree before the key exchange on the parameters, resulting in the same key exchange across all paths but with keys known to the attackers.

Rely on Data Transfer Encryption. Another option is to perform a single DH key exchange on one path (chosen randomly) and then use a multipath data transfer protocol such as MPTCP [1] to send encrypted data. This is practically what would happen if we perform a standard DH key exchange (or any other single path key exchange protocol) on top of MPTCP and then encrypt traffic with the exchanged key.

In this case (see Figure 2, left), an active attacker on the path we performed the key exchange on would have access to the secret keys used by the participants (say k_{AM} and k_{MB}). However, during data transfer, this attacker would use K_{MB} to re-encrypt the data sent by A for B, while the data on the other paths would be encrypted with the key k_{AM} (assuming adversaries cannot communicate – *A/A* – or that the other adversaries are passive – *A-P*). Therefore, at some point B could reject the connection due to the impossibility of decrypting data from these other paths.

While this seems better than DH on a single path, it still exposes the data sent by A to the attacker (note that attackers may collude after the key exchange to gather all the data they collected). Therefore, a straight-forward extension of DH to the multiple path scenario is not enough. This also implies that a straightforward implementation of TLS on top of a multipath protocol such as MPTCP will not increase its security. In Section 10, we show how to combine SMKEX (§5) with TLS to actually increase its security.

[2]The only major change between the original proposal and the current use of the DH protocol, is that in most applications today we use elliptic curves, which add curve points instead of multiplicating integers. This is faster and requires less bandwidth, since the public values are much smaller (e.g. 256 bits for security similar to 2048 bits in the original approach).

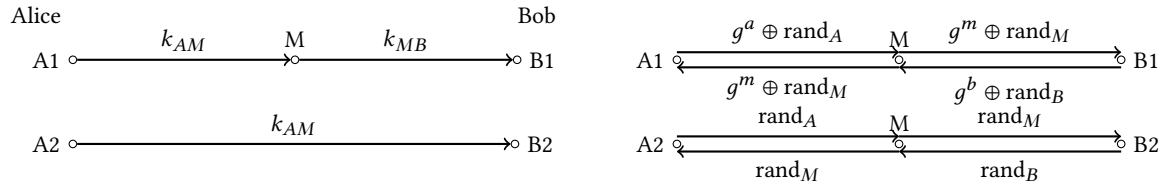

Figure 2: Multipath extensions for Diffie-Hellman. Left: single DH then MPTCP. Right: shared secret Diffie-Hellman.

Shared Secret Diffie-Hellman. Another possible approach is to combine the DH key exchange with secret sharing [45]. This was done by Takano et al. [47] in the context of P2P networks. The idea is to split the public values, e.g. g^a, into shares $s_1, s_2, \ldots$ up to the number of paths, such that $g^a = s_1 \oplus s_2 \oplus \ldots$ (see Figure 2, right). However, both *A-P* and *A/A* adversaries might compromise the exchanged key either during or after the session. For example, let's take the case of *A/A* adversaries that communicate after the session. Say A starts the protocol and sends her shares $s_1, s_2, \ldots$ to B. The attacker M can agree beforehand on the shares he will reply to A ($g^m = s'_1 \oplus s'_2 \oplus \ldots$), without requiring communication during the key exchange. Upon receiving these shares, A creates the shared secret $k_{AM} = g^{am}$ and starts sending data to M (thinking of B). M cannot decrypt the data at this point, since he cannot recover g^a. However, after the protocol run, the attackers can collude to get g^a and finally k_{AM}. At this point it is possible to recover the data sent by A. The same happens on B's side.

Multiple Diffie-Hellman. Yet another simple approach is to perform a different DH key exchange on each communication path, obtaining a different key k^i on each path. Then, A and B can compute a new global key, e.g. by adding the key bits modulo 2: $k_A = k_A^1 \oplus k_A^2 \oplus \ldots$. However, this solution has the same problem as the one above: it does not actually provide a secure key exchange. The attackers can also reconstruct k_A after the key exchange and hence obtain data sent by A.

In summary, none of the simple extensions of DH to multiple paths presented above can provide a secure key exchange against *A/A* and *A-P* adversaries.

5 SECURE MULTIPATH KEY EXCHANGE

In this section, we present our secure multipath key exchange protocol (SMKEX). This protocol provides secure key exchanges with forward and backward secrecy against *A-P* and *A/A* adversaries without relying on long-term secrets or trusted third parties.

SMKEX, shown in Figure 3, performs a standard Diffie-Hellman key exchange on one path while exchanging a hash of session information and some nonces on the other path. Compared to classical Diffie-Hellman, the only extra costs are computing and sending the hash and nonces. As the protocol consists of a single round trip, it can also easily be be included in other protocols, such as the initial TLS key exchange, without introducing additional delays. The protocol fields are described in Table 1.

In isolation, the Diffie-Hellman key exchange on the first path is vulnerable to a man-in-the-middle attack. To prevent this, the server sends a hash of session information (the public keys and the cryptographic nonces) on the second path. The client uses

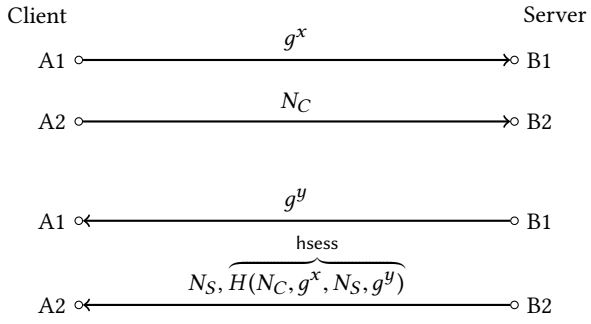

Figure 3: Secure multipath key exchange protocol (*SMKEX*) in the 2-path case.

this additional information to verify that it has the same session information as the server. Additionally, it allows the client to verify that the server's public key was created by the server itself and not chosen by the attackers (because in our model the attackers are unable to synchronize across both paths to forge both the server's public key and the hash).

In the *A-P* scenario, the protocol exploits the fact that the passive attacker is unable to modify messages, so at least one path is safe against tampering. If the Diffie-Hellman exchange happens on this safe path, then the security of the key follows immediately from the properties of the Diffie-Hellman protocol. If the session information hash is sent on the path with the passive attacker, then it will reach the client unchanged. Thus, the client can compare its own session information to what it received from the server and abort the protocol if it detects a mismatch.

For the *A/A* scenario, the protocol exploits the lack of synchronization between attackers. The intuition here is that the attacker that sees the session information hash does not glean any information about the public keys themselves. Due to the random oracle nature of the hash, this makes the attacker unable to construct a new hash that matches the client's own public key and a forged server public key. The client thus either receives an acceptable hash for a session where no man-in-the-middle attempt happened, or a different hash that produces a mismatch and causes the client to abort the session. If the public keys and the nonces match, the client concludes that no man-in-the-middle attack happened and accepts the negotiated secret.

The nonces N_C and N_S could also be sent on the first path, together with the Diffie-Hellman elements. The second path would then only be used for sending the hash. This solution would, however, require the additional assumption that the attacker on the

N_C, N_S	Client and server nonces
g^x, g^y	Client and server Diffie-Hellman public key shares
sess	Session information (nonces and key shares)
hsess	Hash of session information
sk	Negotiated Diffie-Hellman secret
ss	Negotiated secret string
atk	Application traffic key

Table 1: Notation for exchanged messages

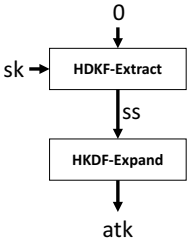

Figure 4: Key derivation tree using HKDF-Extract and HKDF-Expand as defined in RFC5869 [29]. HKDF-Extract with a 0 seed is a good randomness extractor in the random-oracle model.

first path cannot send a message to A2, which is not required by our protocol. The exact security property achieved by SMKEX is described in §6 and in more detail in §A.

We define a session as a tuple containing the client's and server's Diffie-Hellman key shares and hello fields:

$$\text{sess} = (N_C, g^x, N_S, g^y) \tag{1}$$

The value present at the client are indicated via $*$ in the superscript. The session, as viewed by the client, is therefore:

$$\text{sess}^* = \left(N_C^*, g^{x*}, N_S^*, g^{y*}\right) \tag{2}$$

To guarantee session independence, each N_C, N_S, x and y must be randomly generated for each new session.

The client starts the protocol by sending its public value g^x on the first path and its nonce N_C on the second path.

The server uses a randomly generated secret y to compute the shared Diffie-Hellman secret $\text{sk} = g^{xy}$, which is, as we show in §6, a secure Diffie-Hellman key. To extract a key for use in further cryptographic schemes, such as symmetric encryption, one uses a randomness extractor such as HKDF [29], which is depicted in Figure 4. Alternatively, one can use an almost-universal hash function [8] and extract via the Leftover Hash Lemma [24], using N_S as the seed, at the cost of some entropy loss.

The server replies on the first path with its own public value g^y, and sends its nonce N_S and the hash of the entire session information (hsess) on the second path. Upon receiving the two messages, the client computes its own shared Diffie-Hellman secret $\text{sk}^* = g^{xy}$ and extracts a key analogously to the server. The client then hashes its own session information and checks whether the result matches with what it received from the server. If the hashes do not match ($\text{hsess} \neq \text{hsess}^*$), the key exchange fails and the client outputs $\perp$.

While SMKEX provides a clean extension of Diffie-Hellman to multiple *public* channels, its careful design results in better security than the approaches from the previous section: it enables a secure key exchange with forward and backward secrecy for both *A-P* and *A/A* adversaries. Therefore, it can protect even against some categories of active adversaries, increasing the level of protection achievable by opportunistic encryption.

As we describe in §C.2, the protocol can be extended to a larger number of paths at low communication cost: the communication on all further channels is similar to the one on the second channel.

6 FORMAL ANALYSIS

We analyze SMKEX in a model that adapts that of Canetti and Krawczyk [6] to opportunistic multi-path key exchange. We provide a high-level description of our modifications in this section and defer a more formal description of the model to §A.

In the model of Canetti and Krawczyk [6], initiator and responder of a key exchange session obtain as input a *session identifier*. Since the existence of such a predetermined identifier seems unrealistic in opportunistic protocols, we follow the approach of Choo et al. [9] and define the session ID *sid* of a protocol is considered as an output—not an input—of the protocol. To model the property that, in SMKEX, the initiator begins the protocol by sending both messages A1 $\rightarrow$ B1 and A2 $\rightarrow$ B2 as in Figure 3, we let the initiator P_i first be initiated through an invocation $(P_i, P_j, \text{initiate}, id)$ analogously to [6], and subsequently through an invocation $(P_i, P_j, \text{follow-up}, id)$, which lets the initiator send the second message A2 $\rightarrow$ B2 in Figure 3. (The value *id* is only used locally by P_i to identify *which* one of the possibly multiple sessions between P_i and P_j is referred to.) The responder need not receive such an explicit invocation, as the sessions are started by the messages received on the network. In the security game, adversary $\mathcal{A}$ may start an arbitrary number of SMKEX sessions and attack them by observing and modifying network messages, as well as by corrupting parties. At some point, $\mathcal{A}$ selects a test session and obtains *either* the correct key computed in that session *or* a purely random key, and $\mathcal{A}$ must guess whether or not it received the correct key. The advantage $\text{Adv}_{\text{SK}}^{\text{SMKEX}}(\mathcal{A})$ of adversary $\mathcal{A}$ is then defined as

$$\text{Adv}_{\text{SK}}^{\text{SMKEX}}(\mathcal{A}) = 2\Pr\left[\mathcal{A} \text{ guesses correctly}\right] - 1\,,$$

which is analogous to the original model [6].

An adversary for SMKEX can always emulate a client to a server; therefore, the fact that a server session completes and accepts the key although it does not communicate with the actual client is not considered an attack. This resembles the server-only authentication scenario considered in previous work such as by Krawczyk et al. [28], and analogously with that work we require that the test session chosen by the adversary must be a client (or: initiator) session. *A-P* and *A/A* adversaries in the sense of §2 are modeled by restricting the types of adversaries considered in the security definition, as follows.

- *A-P* adversary. The adversary $\mathcal{A}$ delivers at least one of the messages B1 $\rightarrow$ A1 or B2 $\rightarrow$ A2 in the test session

unmodified to P_i. This model also implies that the active attacker cannot inject a message on the path controlled by the passive one, as stated in Section 2.1.

- A/A adversary. The adversary consists of two separate Turing machines $\mathcal{A}_1$ for A1 $\leftrightarrow$ B1 and $\mathcal{A}_2$ for A2 $\leftrightarrow$ B2. (For a precise formal model see §A.) The query $(P_i, P_j, \text{initiate}, id)$ to P_i in the test session, as well as the queries to deliver the messages A1 $\rightarrow$ B1 to P_j and B1 $\rightarrow$ A1 to P_i are made by $\mathcal{A}_1$. The query $(P_i, P_j, \text{follow-up}, id)$ to P_i in the test session is made by $\mathcal{A}_2$, Adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ interact (in the sense of activating, revealing state, or corrupting) with disjoint sets of parties; that is, there is no party P_n such that both $\mathcal{A}_1$ and $\mathcal{A}_2$ make a query targeting P_n.

The above conditions formalize the models described in §2.

In §B.2, we prove the following theorem and show that SMKEX is secure against A-P adversaries if the DDH problem is hard in the considered group. The proof exploits that if A1 $\leftrightarrow$ B1 is attacked passively, then the protocol corresponds to Diffie-Hellman against passive adversaries, and if A2 $\leftrightarrow$ B2 is attacked passively, then the correct transmission of the hash value ensures that a modification to the Diffie-Hellman elements is detected.

The proof, and the subsequent one, are in the random-oracle model. Collision resistance of the hash function is not sufficient for security, which can be seen as follows: In Theorem 2 the hash function is also required to *hide* the input; otherwise the active attacker on B2 $\rightarrow$ A2 could forge.[3] This *hiding* is exactly what we achieve by using the random-oracle assumption. We provide a standard-model construction, based on split-state non-malleable codes, in §D.

Theorem 1. *Let $\mathcal{A}$ be an A-P adversary that makes at most q queries to the random oracle and initiates at most s sessions. Then there is an adversary $\mathcal{B}$, described in the proof, such that*

$$\mathrm{Adv}^{\mathrm{SMKEX}}_{SK}(\mathcal{A}) \leq 2s\mathrm{Adv}^{\mathbb{G}}_{DDH}(\mathcal{B}) + sq/2^{\lambda} ,$$

where $\mathrm{Adv}^{\mathbb{G}}_{DDH}(\mathcal{B})$ is the advantage of $\mathcal{B}$ in distinguishing a DDH triple in $\mathbb{G}$ from a purely random one and λ is the output length of the hash function.

In §B.3, we then show that SMKEX is secure against A/A adversaries, again under the assumption that the DDH problem is hard. The term $stq/2^{2\nu-1}$ corresponds to adversary $\mathcal{A}_1$ guessing both N_C and N_S, mounting a successful man-in-the-middle attack, and $2stq/\#\mathbb{G}$ corresponds to $\mathcal{A}_2$ guessing g^x. The term $st/2^{\lambda}$ comes from correctly guessing the hash value h.

Theorem 2. *Let $\mathcal{A}$ be an A/A adversary that makes at most q queries to the random oracle, initiates at most s sessions at clients and at most t sessions at servers. Then there is an adversary $\mathcal{B}$, described in the proof, such that*

$$\mathrm{Adv}^{\mathrm{SMKEX}}_{SK}(\mathcal{A}) \leq st\mathrm{Adv}^{\mathbb{G}}_{DDH}(\mathcal{B}) + stq/2^{2\nu-1} + 2stq/\#\mathbb{G} + st/2^{\lambda} ,$$

where $\#\mathbb{G}$ is the group order of $\mathbb{G}$.

Both theorems are proven via a sequence of game hops, a standard proof technique. The proofs are deferred to the appendix.

[3] Consider the identity function, which is trivially collision-resistant. Thereby the attacker on B2 $\rightarrow$ A2 learns g^x, and an A/A-attack with predefined g^y will succeed.

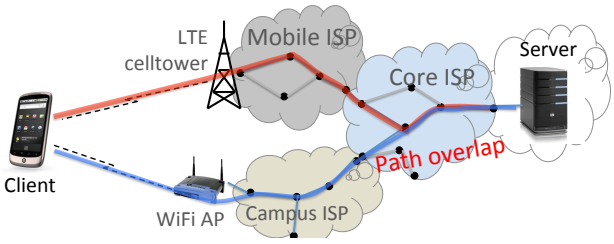

Figure 5: Path diversity available to mobile users.

Since different sessions of the protocol are completely independent, the security statements imply both forward and backward secrecy *between* multiple sessions, but not *within* a session. More formally, the security game allows the adversary to obtain all information except for the ephemeral secrets used in the test session.

7 EMERGING PATH DIVERSITY FOR SECURE MULTIPATH COMMUNICATIONS

The Internet has evolved to the point where it offers physically disjoint paths for many client-server pairs: the hierarchical interconnection between autonomous systems in the Internet is being replaced by a flatter structure where content providers and CDNs peer directly with access networks (DSL, hotspot and cellular), reducing path length and latency and increasing path diversity [31]; content distribution networks deploy servers throughout the Internet and allow content providers to move their data closer to the users; finally, mobile devices, which are one of the main contributors to traffic growth in the Internet [10], connect to the Internet via multiple wireless interfaces such as Wifi and cellular.

To understand the added security provided in practice by SMKEX, in this section we present a brief measurement study of the path diversity available to mobile devices. We have shown that SMKEX is secure in the A/A and A-P settings; to break SMKEX, adversaries must thus be present, active and synchronized across all paths between endpoints, which raises the bar for successful attacks. Consider the example in Figure 5 where the mobile uses Multipath TCP to talk to the server via its two wireless interfaces. In this example, the two paths start out on different networks and converge a few hops away from the server. When the two paths are disjoint, there must be active, synchronized attackers on both paths for a successful attack; in the core operator, however, the attack is simpler because all traffic crosses a single network operator and even a single router.

To estimate the difficulty of executing an attack, we classify attackers based on their ability to subvert one or more Autonomous Systems (or ASes). In this classification, we consider only active attackers, as this is the safer assumption, focusing on the difference between A/A and A-A, since as we mentioned in Section 2 the A-P case collapses to A-A if the active attacker can inject packets into the passive path.[4] Hence, we define the following classes of active attackers, partially inspired from Unger et al. [49, Section III.A]:

[4] *Local* or *nation-wide* MITM that cannot control both paths are not able to establish the A-P setup. Hence, even with the possibility of injecting packets from the active path, they cannot become A-A either. Therefore, the A/A and A-A cases capture the more realistic scenarios to be considered.

Path Overlap	0 AS	1 AS	2 AS	3+ AS
USA	16/70	10/34	0/17	0/26
UK	48/54	26/27	5/13	0/20
Switzerland	30/80	10/75	0/25	0/5
Romania	12/50	26/58	0/30	1/30
Israel	60/60	21/21	4/4	15/30

Figure 6: Path overlap measurements: routes to popular websites are surprisingly disjoint, with many having no overlap regardless of the origin country.

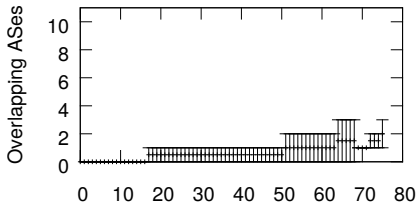

Figure 7: AS path overlap when a mobile client uses a cellular and a fixed connection (USA)

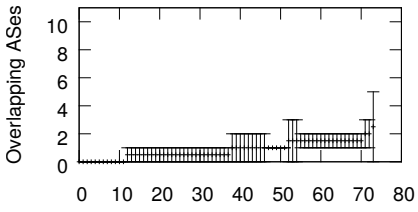

Figure 8: AS path overlap when a mobile client uses a cellular and a fixed connection (Romania)

- *local MITM*: an attacker controlling local networks (e.g., owners of Wifi access points or localized internet service providers).
- *nation-wide MITM*: an attacker controlling small parts of the Internet, such as the internet service providers (ISPs) of a country or small geographical regions.
- *global MITM*: an attacker controlling large segments of the Internet, such as powerful nation states or large ISPs.

7.1 Measurement study

We set out to measure the path diversity that exists in practice for dual-homed mobile clients. As servers, we used Alexa's top 100 websites to which we added some sites such as local newspapers (which may be subject to surveillance or censorship). We ran traceroute from client devices connected via a variety of fixed and mobile networks in countries where we had access to mobile clients (via volunteers): USA, UK, Switzerland, Romania and Israel. While our study is by no means exhaustive, it does shed a light on the amount of path diversity that exists today in the Internet. For each traceroute, we first perform a DNS lookup and then traceroute to the resulting address. This means that our traceroutes may be redirected to different servers serving the same website.

A previous study has tried to examine the same question starting from an inferred AS-level map of the Internet [38], estimating *offline* the amount of path diversity there exists between any two endpoints. The study's conclusions paint a mixed picture, noting that *"only about 5% of the countries show good chances of being robust against MITM from a device view"*, however they note that *"careful choice of the edge providers could make this likelihood positive for a majority of the countries"*. Our study is complementary: it only focuses on a subset of the Internet (a few edge ASes and the top 100 servers) but it focuses on mobile devices and it is much more accurate, because it uses actual Internet routing (as opposed to estimated routes based on the AS graph) and because it also measures the effect of CDNs on path diversity.

The basic metric we are interested in is *path overlap*: the number of autonomous systems that are traversed by *both* the mobile and wired path en-route to the server. We do not count the destination AS as path overlap: if an attacker controls the server (or its AS) there is little SMKEX can do.

To measure path diversity, we traced the paths from all or a subset of the cable providers and mobile operators in each country to our target websites (2-5 fixed operators and 1-5 mobile ones per country). We then studied the path overlap when the client uses any cable operator in conjunction with any mobile operator in their country. When traceroutes include private addresses we assume these belong to AS0; thus if AS0 appears in multiple traceroutes, we assume the paths overlap, when in practice they might be different.

As expected, our experiments show that AS path lengths to Alexa top 100 sites are short: 40% of paths have three AS hops or less, and 95% have six hops of less; the longest path contains 11 ASes.

Measurement results in Table 6 show the minimum and maximum path overlap across all mobile-fixed provider combinations to the target websites that respond to traceroute, per country. Depending on the choice of providers, as many as 50 to 80 (out of 100) websites can be reached via AS paths without any overlap, or 12 to 50 in the worst fixed/cellular provider combination. Most remaining websites can be reached via paths that have only one AS in common. In the best case, none of the paths to our targets had overlaps including two or more hops for most countries. Israel is a special case because we only had access to one mobile operator and one cable operator - even so, 80 websites can be reached via paths that have at most one AS hop in common.

In Fig. 7 and 8, we show the measurements for USA and Romania in detail. Romania has the highest path overlap of all the countries we studied; still, if one chooses the best fixed-mobile provider combination, there is no path overlap for 50 websites, and a single AS is present on the two paths for the remaining 25 servers in our set. In the USA, almost 70 websites can be reached without any overlap, and all that respond to traceroutes can be reached with paths that overlap in at most one AS. Note that for these two countries, only 75 of the 100 websites can be traced via all the networks we used. This is because ICMP TTL exceeded messages are filtered on at least one path. We omit these servers from our detailed analysis, and in the summary results shown in Table 6, we assume that they are a worst case that have three or more overlapping AS.

These results are very encouraging: for most destinations no single AS, including cable, mobile or transit operators, can by itself mount MITM attacks against dual-homed mobile clients using SMKEX. This means that SMKEX is secure against local MITM for most of Alexa's top 100 websites in the countries we have studied.

On the downside, the Alexa top 100 sites we traced are likely better connected at the AS-level than less popular destinations, so these results might paint a rosier picture than reality. To understand the level of overlap we might see for other less popular websites, we examined connectivity to Akamai, the largest CDN in operation,

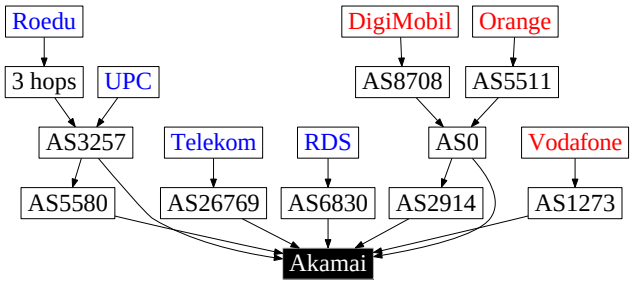

Figure 9: Routes to Akamai from endpoints in Romania.

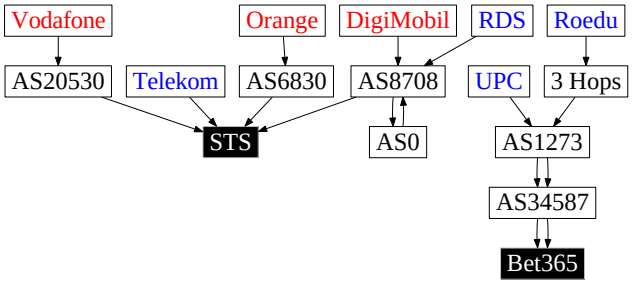

Figure 10: Routes to Bet365 from endpoints in Romania.

and show the results for Romania (where path diversity is worst among the countries we measured). While large content providers are building their own distribution networks (e.g. Google, Facebook and Microsoft), most other content providers are turning to Akamai (or other CDNs) to help them fight DDoS attacks and terminate TLS sessions close to the customers. Paths to Akamai are therefore a proxy for path diversity in this large category of websites. In Figure 9, we show paths from Romanian clients to Akamai. In the figure, the operators shown in red offer mobile access, and the ones in blue offer fixed connectivity. Paths to other major providers (Google, Microsoft) are similar (ommited for brevity). The figure shows very encouraging results: to successfully attack SMKEX when communicating with Akamai from a single fixed operator, a nation-wide attacker must intercept traffic flowing via that operator and all possible secondary paths, such as the three mobile operators shown in red. Therefore, in our experiments, SMKEX is secure against local MITM adversaries also for websites using Akamai.

These results show that SMKEX can raise the bar for successful attacks against opportunistic encryption beyond the reach of local MITM to nation-wide MITM attackers in many practical situations.

7.2 Protection against nation-wide attackers

We discovered an interesting example of nation-wide attack behavior in the routes towards *bet365.com*, a betting website hosted in Hong Kong. The routes we measured are shown in Figure 10. We see that most clients are redirected to STS (special telecommunications service of the Romanian Government) while clients using Roedunet or UPC reach the actual website in Hong Kong. In fact, all other clients get a page saying that traffic is restricted to Bet365 because it does not comply with Romanian law. Detailed analysis has shown that the redirection is achieved via static DNS routes to STS servers returned by the operators' DNS resolvers; if we switch the resolver to a public one (e.g. Google's 8.8.8.8), all clients resolve Bet365 to the correct address, and can reach the website: IP traffic is not restricted at all. Note that both the DNS resolvers of Roedu, the educational network operator, and that of UPC resolve Bet365 correctly; we could not explain the reason for this different behavior.

To protect against such nation-wide attackers, in our example above it is sufficient to use a path through one of the operators that was not rewriting DNS; but this will not work against more diligent states. We propose a general and pragmatic solution to create artificial path diversity that relies on tunneling. Before any communication is made, all clients set up long-lived tunnels that cross jurisdictional and geographical boundaries, and all servers use a CDN that has a international footprint (i.e. deployments in multiple countries).

To set up such tunnels, clients can rely on cloud computing and rent virtual machines in other countries, as shown in Figure 11, where a user based in Europe sets up a long-lived tunnel between his machine and its VM in the US. This user will have IP address A1 in Europe, and address A2 in the USA as provided by the cloud provider. The tunnel is secured when the user first registers with the cloud, and will be used to create path diversity for *all connections* this user makes with other parties[5]. The benefits of this setup are clear: the *public* path segments are *small* and completely disjoint (in Europe and the US). Furthermore, only SMKEX setup messages need to cross the Atlantic, all data traffic can stay on the European path, avoiding unnecessary costs. To achieve this, the client can simply close the trans-Atlantic subflow after SMKEX negotiation finishes successfully.

To understand how well such a solution may work, we rented virtual machines in multiple Amazon EC2 datacenters worldwide, and ran measurements from these VMs to our Alexa top 100 servers list. We analyzed AS path overlap between these routes and our local ones, finding that there is little overlap in general. However, disjoint AS paths are not enough to ensure protection against nation-wide attackers (e.g. governments of the countries traversed by our traffic).

Ideally, we would place the traceroutes on the world map, and examine whether there is any country that is crossed by both the Amazon and the local (e.g. Romanian) path. Unfortunately, using IP geolocation to map the routes yields big errors. Even worse, fiber path layouts are not well known or easily traceable.

We use, instead, the idea of Alibi routing[34] that uses speed of light as proof a certain path does not visit a remote country. This concept is shown in Figure 11: if the sum of the RTTs measured between $A_1 - B$ and $A_2 - B$ is smaller than the time it takes light to travel from A_1 to A_2 and back, then the two paths cannot be overlapping anywhere. Since we know the location of our client and our datacenter (as reported by the cloud operator), we can compute a lower bound beyond which it is impossible for the two paths to overlap. In particular, it is impossible for nation-wide attackers situated in Romania or USA to see both paths and break SMKEX.

[5]This is the only time the client will need to perform such setup. This can be done by using SMKEX with many public channels – to reduce possibility of synchronized attacks –, TLS with one ore more trusted certificates, a quantum key exchange, visiting the remote site in person, or any other method that is deemed secure by the client, since this secure connection only needs to be established a *single* time, enabling the use of SMKEX for securing the communication to any other website.

Datacenter	USA			UK			Switzerland			Romania			Israel		
	Thresh	*Mobile*	*Fixed*	*Thresh*	*Mobile*	*Fixed*	*Thresh*	*Mobile*	*Fixed*	*Thresh*	*Mobile*	*Fixed*	*Thresh*	*Mobile*	*Fixed*
Virginia	N/A	N/A	N/A	60ms	3%	48%	68ms	0-50%	50-66%	82ms	34-48%	47-65%	97ms	24%	87%
Frankfurt	68ms	30-32%	39-56%	5ms	0	6%	8ms	0	0-15%	17ms	0-10%	6-13%	28ms	0	70%
Sydney	121ms	34-43%	38-46%	170ms	11%	50%	165ms	41-50%	51-68%	152ms	42-50%	48-68%	141ms	44%	76%

Table 2: Using geographic diversity to ensure security: for different country and location of our tunnel, we list the percentage of Alexa top 100 websites to which the measured RTT across the "public" paths is smaller than the theoretical minimum RTT between the two paths.

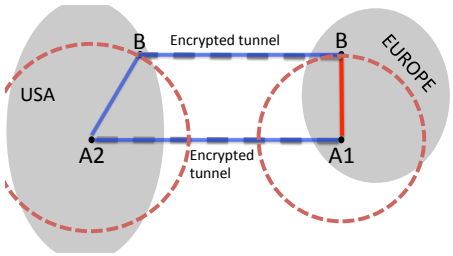

Figure 11: Using long-term tunnels to ensure path and jurisdiction diversity.

We present our results, for three Amazon datacenters (Frankfurt, Virginia and Sydney) and clients in five countries in Table 2. For each country / datacenter combination, we compute the theoretical latency threshold by dividing the geographical distance between the origin country and the country hosting the datacenter to the speed of light in fiber. Then, we used the measured latency to Alexa top 100 websites from various vantage points. If the added RTT from the client to website *A* and from the datacenter to website *A* is smaller than this threshold, then the paths are provably disjoint and are safe against one nation-state attack.

In the table, we split our results based on the type of network operator (fixed/mobile) and show the minimum and maximum fraction of Alexa websites for which there exist such secure paths.

The table shows that a good fraction of websites can be reached by paths that are secure against one nation state, but this percentage depends on a few factors. First, when the datacenter is close to the origin country (e.g. Frankfurt for Switzerland, UK or Romania), there are very few sites with safe routes; this is because the geographical distance is small, resulting in a small RTT; on such distances, the router and server processing times affect the measured RTTs considerably.

When we use VMs further away (e.g. USA), the fraction of sites reachable via guaranteed non-overlapping paths by European clients increases to 50%, which is quite remarkable and is explained by the fact that all these sites have local replicas very close to the clients. Non-replicated sites provide no guarantees of path disjointness. Finally, using a tunnel to Sydney only marginally improves the results; this is because almost all replicated sites were already "covered" by the US datacenter.

Another observation is that, in all our datasets, the fraction of secure paths when using mobile operators is smaller than for fixed operators. This is expected, since wireless latencies (e.g. LTE or 3G) are known to add at least 20ms to the wired RTT for any given destination.

These results show that SMKEX can also protect against nation-wide attackers for a majority of popular websites, if we allow the use of an encrypted tunnel between continents.

8 IMPLEMENTATION

Mobile devices are switching to Multipath TCP (MPTCP) [1], a recently standardized TCP extension that can utilize multiple paths (called subflows) within a single transport connection. Past research has shown that MPTCP can be used to ensure smooth mobility between cellular and Wifi networks [43] or between overlapping Wifi deployments [12]. MPTCP has already been widely deployed on mobile phones on all Apple devices and top-end Android devices (such as the Samsung Galaxy 7 & 8 series).

Hence, to make experimentation with SMKEX simple for users and to increase its impact, we have implemented SMKEX over MPTCP. Our implementation has two main parts: a) the SMKEX library, running in user-space that allows unmodified applications to use our opportunistic encryption, and b) the integration with a Multipath TCP Linux kernel implementation, including some minor kernel changes.

User-space Library. Our user-space library allows apps that rely on the TCP sockets API to run over SMKEX/MPTCP without any changes. To this end, we overwrote the socket API calls and packaged our algorithms as a dynamic library which can be pre-loaded at program instantiation time before `libc`; this way unmodified apps will use our implementations of the socket APIs instead of the system implementations. Our library code relies on the system calls to interact with the MPTCP implementation. We implemented all the cryptographic operations using the OpenSSL crypto library. For the key exchange part, we used 256-bit ECDH keys and SHA-256 hashes, and AES-GCM for data transfer experiments.

MPTCP integration. Integrating SMKEX with MPTCP is in principle straightforward, however there are some subtleties to properly ensure path diversity and to ensure resilience to DNS hijack attacks, which we discuss in greater detail next.

An MPTCP connection contains one or more subflows, and it starts when its first subflow is created. Each subflow looks very much like an independent TCP connection to the network, with the exception that its segments carry MPTCP-specific options. After the initial subflow is set up, each endpoint computes an MPTCP token, which is a unique identifier its peer has assigned to this connection. This token is embedded in the handshake of additional subflows within the same MPTCP connection and helps the remote end find the appropriate connection to bind the subflow to. Secondary subflows cannot be set up until the initial subflow has been set up.

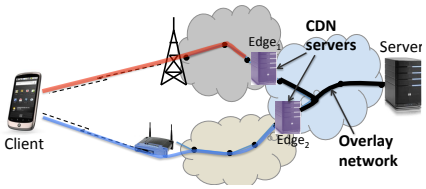

Figure 12: CDNs terminate TCP close to the end user.

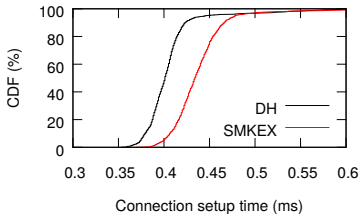

Figure 13: CDF of connection setup time, RTT=0.2ms.

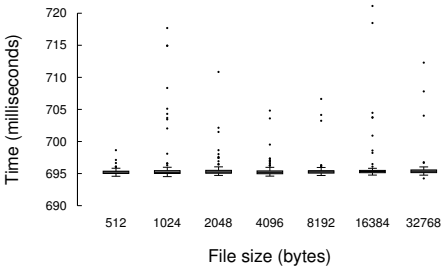

Figure 14: Data transfer duration using SMKEX over trans-atlantic tunnels.

To run SMKEX, our library implementation first opens an MPTCP connection and uses an MPTCP API (from Hesmans et al.[23]) that blocks until the specified number of subflows is created. If the connection falls back to regular TCP the SMKEX handshake fails; the same happens if the desired number of subflows is not created in a predefined amount of time.

Once the MPTCP connection has enough active subflows (two by default), SMKEX can start the handshake. However, if we simply send SMKEX messages using the sockets API provided by default by MPTCP, there is no control over the subflow that will carry the handshake data. In most cases, all the messages will be delivered by the first subflow, and this would break the security of SMKEX making MITM attacks easy to execute.

To send and receive data on specific subflows, we have made two changes to the MPTCP Linux kernel implementation as follows: a) the `send` syscall allows specifying which subflow must carry the provided application data, and b) the `receive` syscall allows specifying which subflow to receive data from. When data is received on another subflow than the expected one, the `recv` returns a specific error code telling the library which subflow it should read data from. Finally, to avoid changing the syscall API, our implementation reuses an unused byte in the `flags` parameter to specify the desired subflow in the `send` and `recv` calls.

Finally, we use the *fullmesh* MPTCP path manager that, by default, creates one subflow for each interface: a mobile client, for instance, will create two subflows to the server, one on cellular and one on Wifi.

CDN integration. There are two deployment scenarios we target; in both scenarios, a server and a mobile client support SMKEX running over MPTCP. The first scenario is the one depicted in Figure 5, where the MPTCP subflows are terminated at the server. The second scenario involves a CDN and is shown in Figure 12: in this case the two subflows do not reach the same server, and an additional mechanism is required to direct the secondary MPTCP subflows to the appropriate server. This is the preferred scenario for SMKEX because it provides the best path diversity, as shown in our measurement study; we discuss it next.

To route client traffic to nearby edge servers, CDNs use one of two approaches: DNS redirection, where the client location is used to select a local replica, or use IP anycast where all edge servers advertise the same IP address and Internet routing distributed clients to their closest servers. In this paper, we assume the edge servers rely on IP anycast; this solution is used by many, including the Microsoft CDN [21].

Consider the example in Figure 12, where the service address *A* is advertised in BGP by both edge servers. When the MPTCP connection starts over the cellular interface, its first subflow will be handled by edge server 1 which is closest to the client (from a routing hops point of view). Edge server 1 will serve content from its local cache, or contact the origin server the required content is not cached. When the client opens its wireless subflow, the resulting subflow will reach edge server 2. The only remaining problem is that edge server 2 must now forward the subflow to edge server 1 over the CDN's internal network.

To achieve this, we use Beamer [40], a load balancer that supports MPTCP. To use Beamer, the CDN first assigns a unique numeric identifier to each of its edge servers. Beamer works as follows: when the first subflow is setup to edge server 1, the edge server will tell the client its unique identifier. The client will include this identifier into its second subflow which will reach edge server 2; this server will simply proxy the connection to edge server 1.

Note that on-path attackers can modify the connection ID, but the only effect is that the secondary subflows will be rerouted incorrectly in the CDN network, and will be broken (i.e. edge server 2 will send the subflow to another edge server instead of sending to edge server 1). In this case, the SMKEX handshake will fail.

Practicality of SMKEX. While this section shows that SMKEX relies on CDN's and popular websites to update their infrastructure in order to ensure the highest security possible, it is important to note that MPTCP deployment requires the same changes as SMKEX: an MPTCP enabled kernel and a load balancer. We believe such deployment is near because load balancers are already widely deployed in production [18, 36, 41] and MPTCP is already widely deployed on mobile clients.

9 EVALUATION

The goal of our evaluation is to test the correctness of our implementation and its behavior in practice. We tested our implementation on our local testbed and using Amazon to create wide-area path diversity.

Testbed experiments. In our first experiment, the client and server run on two quad-core Xeon machines connected via two Gigabit links emulating the different paths. Our client repeatedly sets up an encrypted connection to the server and we measure the time it takes to perform the connection handshake. Figure 13 shows the

CDF of connection setup times for SMKEX compared to standard Diffie-Hellman. In the median, SMKEX takes about 50μs more than standard Diffie Hellman; this difference is explained by the additional round-trip time our MPTCP-based implementation requires. MPTCP only sets up the second subflow after the first subflow is setup.

Amazon EC2 Experiments. To test our path diversity setup, we rented two VMs in two EC2 datacenters on the east coast (Virginia and Ohio). One VM terminates a long term client tunnel, offering path diversity. The other VM is used to emulate an edge CDN server.

Our server and client are close to each other (5ms RTT), but they also set up a path via the USA using long-term openVPN tunnels to one of the Amazon VMs. Our client repeatedly downloads files of different sizes from the server. In figure 14 we plot the total download time. The measured latencies are as follows: the long path has an RTT of 280ms (crossing the Atlantic four times), and the short path has an RTT of 5ms.

For small files, the expected download latency should be dominated by the long path RTT: our implementation requires two RTTs over this path, one to setup the MPTCP subflow and one to perform the key exchange. After the key is set up, the server sends all data via the low latency path. The experiments confirm this hypothesis the latency is around 650ms for all file sizes we tested. The file size has little influence on the download because the local, high-speed (50Mbps) link is used for data transfer.

10 EXTENSIONS OF SMKEX

Due to its simplicity, SMKEX may be easily extended to actually increase the security of TOFU or TLS-like protocols. In the previous sections, we have focused on the basic version of SMKEX, because this provides the highest degree of usability. As Unger et al. [49] write, *"defending against mass surveillance requires a communication system that virtually all users can successfully use. Thus, it may be wise to start from the basic user experience of today's widely deployed communication apps and try to add as much security as possible..."*. Hence, in this section we show two possible enhancements of SMKEX: (*a*) TOFU enhancement; (*b*) TLS integration. In §C.3, we also discuss the possibility of using double ratcheting to provide forward and backward secrecy even across messages from a single session. Note that these extensions are shown informally, with the goal to show how SMKEX can be used in various other scenarios. We leave a formal analysis of these extensions and their implementation to future work.

10.1 TOFU enhancement

SMKEX can be enhanced by using a Trust-on-first-use (TOFU) approach[6], increasing the security of applications that rely on TOFU authentication, such as SSH or websites using self-signed certificates.

This could be done by having the server use a long-term public/private key pair which is stored by clients and then used together with the server's ephemeral public key to derive the session key. That is, if we let $X = g^x$ be the ephemeral public key of the client (with corresponding private key x), $Y = g^y$ be the ephemeral public key of the server (with private key y) and $L_S = g^{ls}$ be the long-term public key (with corresponding private key ls), then the client can send as input to the key derivation the concatenation of Y^x and L_S^x, while the server would use X^y and X^{ls}. A similar approach, but requiring the client to also use a long-term key, is used in Signal [46], known as triple Diffie-Hellman, possibly inspired from Protocol 4 of Blake-Wilson et al. [5]. A similar protocol has also been proposed and analyzed by Krawczyk and Wee [30].

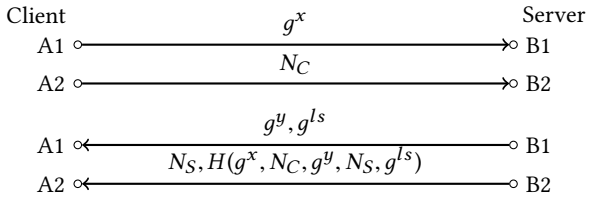

Figure 15: TOFU-based *SMKEX* protocol.

A depiction of our modified protocol with the server using a long-term key is shown in Figure 15. As mentioned by Wendlandt et al. [50], existing TOFU protocols suffer from two main issues: a) possible active attacks during the first connection; b) possible active attacks during an update of the server's long-term key. With our TOFU-based SMKEX protocol, such attacks are no longer possible in the *A*/*A* and *A*-*P* scenarios. Therefore, by storing and checking the server's long-term public key, our TOFU-based SMKEX protocol provides partial protection against active adversaries (i.e., *A*/*A* and *A*-*P*) during initial setup and during server key update (which was not the case for previous TOFU approaches), while providing protection even against *A*-*A* adversaries if these are not able to synchronize during the initial key setup (or key update).

The main disadvantage of TOFU-based approaches (including this extension of SMKEX), is that when long-term keys change (either genuinely or due to an attack), the client is forced to either: *a*) drop the connection (if we want no user interaction) or *b*) ask the user about what to do in this case (which might hinder usability).

Nevertheless, given that previous TOFU-based methods provided the highest security for opportunistic encryption (see the survey of Unger et al. [49, Table I]) and that SMKEX also increases the security of TOFU approaches, we can conclude that *using a TOFU-enhanced SMKEX protocol provides the highest security for opportunistic encryption to date.*

10.2 Integration into TLS

We can also easily integrate SMKEX with TLS, obtaining a combined protocol (which we called MTLS) that provides increased security over TLS, while retaining all the security benefits of the classic single-path TLS. This extension benefits TLS security in two ways: first, it provides improved opportunistic security to unauthenticated TLS, which is described in [44, Section C.5]. Second, it works as an additional barrier in case of Certificate Authority (CA) attacks, as described below.

Several attacks on TLS have exploited problems with CAs: some have issued certificates to invalid parties [33], some have been attacked and rogue certificates issued [17, 20], checking revoked certificates is difficult [35], many share their secret keys with possibly less-secure partners [7]. Privacy issues also appear when large institutions monitor employees with the help of fake certificates

[6]Term probably coined by Wendlandt et al. [50].

Protocol	*A-P*, *A*/*A*		*A-A*	
	Auth	Rogue	Auth	Rogue
SMKEX	✓	-	X	-
TLS	✓	X	✓	X
MTLS	✓	✓	✓	X

Table 3: Comparing the security features of SMKEX, TLS and MTLS

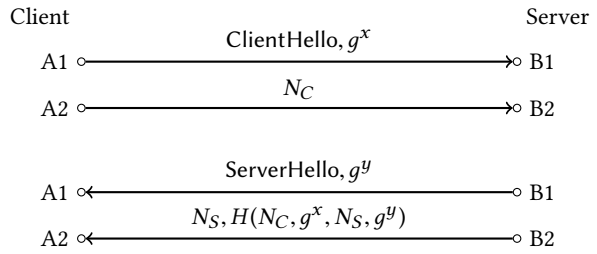

Figure 16: Key exchange in Multipath TLS (*MTLS*) protocol. The first path executes the standard TLS key exchange, while the second path is used to validate keying information similarly to SMKEX.

installed in browsers. Finally, governments might force their ISPs and local CAs to collaborate and trick users into using rogue certificates. By combining SMKEX with TLS we may thwart such attacks. We illustrate the advantage of MTLS over single TLS or SMKEX in Table 3 for the different attackers (*A-P*, *A*/*A*, *A-A*) and scenarios (authentic and rogue certificates).

We illustrate our design using TLS1.3. SMKEX is supported with the (EC)DHE exchange mode and the PSK with (EC)DHE key exchange mode. Accordingly, we assume that the client and servers exchange some form of Diffie-Hellman public key shares g^x and g^y (finite-field or elliptic curves). We only improve the security of the server authentication portion of TLS; client authentication and other key exchange extensions downgrade to the single path case. Figure 16 illustrates the MTLS key exchange. The standard TLS key exchange runs on the first path, with two modifications. First, the Client Hello message indicates in an extension that MTLS is used, and the ClientHello.random N_C in this message, as well as the ServerHello.random N_S in the response, are dropped.[7] MTLS introduces two new messages on the second path. The client sends N_C, and the server responds with N_S as well as a hash of N_C, N_S, g^x and g^y.

MTLS provides all the security of a standard TLS exchange, with added protection against attackers that forge the server's long term secret. For example, in the case of a forged certificate an attacker that is only present on the first path is unable to successfully complete the key exchange with the client. The verification of the hash fails at the client, and the key exchange terminates immediately.

More precisely, MTLS provides security against all attackers for which the test session is either fresh according to the original definition [6], i.e. the server session is uncorrupted, or the adversary behaves as an *A*/*A* adversary, in which case it can even corrupt the (long term key of the) server session.

Theorem 3 (MTLS; informal). *For the described type of adversary, MTLS is secure under the condition that the DDH assumption holds in the selected group and that the signature scheme used in TLS is existentially unforgeable. The statement holds in the random-oracle model.*

The proof is essentially a combination of the proof of Theorem 2 and a standard analysis of the TLS 1.3 core protocol.

11 RELATED WORK

There have been many schemes proposed for trust establishment. We refer the reader to the survey of Unger et al. [49, Section III] for a comprehensive treatment. Here, we shall focus mostly on opportunistic approaches, popular protocols such as TLS, QUIC and Signal, as well as some previous proposals for using multiple channels to establish trust.

One of the best-known protocols for basic opportunistic encryption is Tcpcrypt [4]. It performs an efficient key exchange[8] over a TCP connection to derive keys and then can output an authentication tag over the session transcript, which could be verified using a trusted certificate or using a different communication channel. As in SMKEX, we could send the authentication tag over a secondary public channel (secondary subflow in MPTCP) rather than relying on a certificate or different form of communication channel. Hence, our design can also be seen as a model for increasing the security of previous opportunistic encryption methods such as Tcpcrypt.

An enhanced version of basic opportunistic encryption is done through trust-on-first-use (TOFU), as is the case for SSH or using self-signed certificates, where the client remembers the first long-term key sent by the server. However, as Wendlandt et al. [50] mention, such approaches are completely vulnerable to active MITM attackers during the initial key setup or during key update. SMKEX can protect against many active attackers at all all times. Hence, by combining SMKEX with TOFU, we obtain the best protection possible to date for opportunistic encryption.

The most popular protocols for securing client-server communications, TLS [13] and QUIC [22], as well as the most popular protocol for secure messaging, Signal [46], all rely on trusted third parties to issue correct certificates (TLS, QUIC) or long-term public keys (Signal). However, such trust is problematic, as third parties can become corrupted [17, 20, 33], verification is difficult [35] and keys may be shared among many untrusted parties [7]. Hence, several schemes have been proposed to cope with these issues (mainly focusing on certificates), including: *a*) monitoring issued certificates [19, 50]; *b*) creating and managing public logs of all issued certs [32]; *c*) proposing modifications to the existing architecture [26, 27]. Unfortunately, all of these schemes still require trust in one or more entities. Furthermore, solutions in the first category add overhead to TLS connections, the solutions in the second category cannot quickly cope with compromised private domain keys, while those in the third category remain largely impractical due to the many actors and work required.

[7] To keep the message format compatible with standard TLS, the random fields can be replaced by independent values and ignored in the computation.

[8] Shifts the expensive part of public-key encryption to the client in the case of key exchange based on RSA encryption of a fresh symmetric key, reducing burden from the server and encouraging wide adoption of encryption.

Finally, there have also been some proposals for using multiple communication channels. Some of these require using a secure channel [52], while others propose to use secret sharing for distributing the key [47], which we have shown not to be secure in our setting (see §4). In contrast, SMKEX provides a *secure* key exchange, proven against *A-P* and *A/A* adversaries, which works across *public* channels.

12 CONCLUSION

SMKEX allows the most secure opportunistic encryption method to date, by relying on several *public* communication channels. We have proven that it provides secure key exchanges with forward and backward secrecy across a wide range of adversaries and we have shown that the current path diversity across the Internet allows SMKEX to protect against local and nation-wide active man-in-the-middle attackers. Its simplicity also means that we can easily integrate SMKEX with TOFU and TLS-like protocols, increasing their security.

We have a fully-working implementation of SMKEX, based on a modified Linux MPTCP kernel implementation and a user-level library, which allow unmodified applications to use SMKEX. Therefore, SMKEX is ready to be used. Servers only need to support MPTCP across their edge servers, which can be easily done with the methods we have shown. With the ongoing deployment of MPTCP, we expect this to happen soon, but perhaps the advantages of SMKEX can motivate some of the large web sites to deploy MPTCP even sooner.

SOURCE CODE

The source code for SMKEX is available here:
https://github.com/nets-cs-pub-ro/smkex
and the modified MPTCP kernel, required for SMKEX, is here:
https://github.com/nets-cs-pub-ro/mptcp-smkex

Acknowledgement:

We thank all the people that have helped us in this work, through ideas, experiments, comments in previous drafts and several other ways. Among them, we thank Ross Anderson, Virgil Gligor, Markus Kuhn, Mike Bond and Hugo Krawczyk.

This work was sponsored in part by the European Commission, through the SSICLOPS H2020 project.

REFERENCES

[1] A. Ford and C. Raiciu and M. Handley and O. Bonaventure. RFC6824:TCP Extensions for Multipath Operation ... https://tools.ietf.org/html/rfc6824.

[2] D. Aggarwal, S. Agrawal, D. Gupta, H. K. Maji, O. Pandey, and M. Prabhakaran. Optimal computational split-state non-malleable codes. In *Theory of Cryptography*, volume 9563 of *LNCS*, pages 393–217, 2016.

[3] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. R. Stinson, editor, *Advances in Cryptology — CRYPTO 1993*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993.

[4] A. Bittau, M. Hamburg, M. Handley, D. Mazieres, and D. Boneh. The case for ubiquitous transport-level encryption. In *USENIX Security Symposium*, pages 403–418, 2010.

[5] S. Blake-Wilson, D. Johnson, and A. Menezes. Key agreement protocols and their security analysis. *Crytography and Coding*, pages 30–45, 1997.

[6] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 453–474. Springer, 2001.

[7] F. Cangialosi, T. Chung, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. Measurement and analysis of private key sharing in the https ecosystem. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 628–640, New York, NY, USA, 2016. ACM.

[8] L. Carter and M. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[9] K.-K. R. Choo, C. Boyd, and Y. Hitckcock. Examining indistinguishability-based proof models for key establishment protocols. In *Advances in Cryptology — ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 585–604. IACR, Springer, 2005.

[10] Cisco. Global Mobile Data Traffic Forecast. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html.

[11] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the signal messaging protocol. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*, pages 451–466. IEEE, 2017.

[12] A. Croitoru, D. Niculescu, and C. Raiciu. Towards wifi mobility without fast handover. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 219–234, Oakland, CA, 2015. USENIX Association.

[13] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol version 1.2. RFC 5246. 2008.

[14] W. Diffie and M. Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[15] V. Dukhovni. Opportunistic Security: Some Protection Most of the Time. 2014.

[16] S. Dziembowski, K. Pietrzak, and D. Wichs. Non-malleable codes. In *ITCS*, 2010.

[17] Eckersley, P. Iranian hackers obtain fraudulent HTTPS certificates: How close to a Web security meltdown did we get? https://www.eff.org/deeplinks/2011/03/iranian-hackers-obtain-fraudulent-https. Last accessed: November 2017.

[18] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. das Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein. Maglev: A fast and reliable software network load balancer. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 523–535, Santa Clara, CA, Mar. 2016. USENIX Association.

[19] Electronic Frontier Foundation. Ssl observatory. https://www.eff.org/observatory.

[20] Fisher, D. Final report on diginotar hack shows total compromise of ca servers. https://threatpost.com/final-report-diginotar-hack-shows-total-compromise-ca-servers-103112/77170/. Last accessed: November 2017.

[21] A. Flavel, P. Mani, D. Maltz, N. Holt, J. Liu, Y. Chen, and O. Surmachev. Fastroute: A scalable load-aware anycast routing architecture for modern cdns. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 381–394, Oakland, CA, 2015. USENIX Association.

[22] R. Hamilton, J. Iyengar, I. Swett, and A. Wilk. QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2. 2016.

[23] B. Hesmans, O. Bonaventure, and F. Duchene. A socket api to control multipath tcp (draft-hesmans-mptcp-socket-03). 2008.

[24] R. Impagliazzo, L. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *STOC*, 1989.

[25] R. Joyce. Disrupting nation state hackers. San Francisco, CA, 2016. USENIX Association.

[26] T. H.-J. Kim, L. Huang, A. Perrig, C. Jackson, and V. Gligor. Transparent key integrity (tki): A proposal for a public-key validation infrastructure. *Technical Report CMU-CyLab-12-016, Carnegie Mellon University*, 2012.

[27] T. H.-J. Kim, L.-S. Huang, A. Perrig, C. Jackson, and V. Gligor. Accountable key infrastructure (aki): A proposal for a public-key validation infrastructure. In *Proceedings of the 22nd international conference on World Wide Web*, pages 679–690. ACM, 2013.

[28] H. Krawcyzk, K. G. Paterson, and H. Wee. On the security of the TLS protocol: A systematic analysis. In *Advances in Cryptology — CRYPTO 2013*, Heidelberg, 2013. Springer.

[29] H. Krawczyk and P. Eronen. Hmac-based extract-and-expand key derivation function (hkdf). 2010.

[30] H. Krawczyk and H. Wee. The optls protocol and tls 1.3. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 81–96. IEEE, 2016.

[31] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian. Internet inter-domain traffic. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 75–86, New York, NY, USA, 2010. ACM.

[32] B. Laurie, A. Langley, and E. Kasper. Certificate Transparency. 2013.

[33] Lemos, R. Microsoft warns of hijacked certificates. https://www.cnet.com/news/microsoft-warns-of-hijacked-certificates/. Last accessed: November 2017.

[34] D. Levin, Y. Lee, L. Valenta, Z. Li, V. Lai, C. Lumezanu, N. Spring, and B. Bhattacharjee. Alibi routing. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 611–624, New York, NY, USA, 2015. ACM.

[35] Y. Liu, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. Maggs, A. Mislove, A. Schulman, and C. Wilson. An end-to-end measurement of certificate revocation in the web's pki. In *Proceedings of the 2015 Internet Measurement Conference*, pages 183–196, New York, NY, USA, 2015. ACM.

[36] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 15–28, New York, NY, USA, 2017. ACM.

[37] MPTCP Blog. Commercial usage of multipath tcp. http://blog.multipath-tcp.org/blog/html/2015/12/25/commercial_usage_of_multipath_tcp.html. Last accessed: November 2017.
[38] D. Nguyen, D. C. Phung, S. Secci, B. Felix, and M. Nogueira. Can MPTCP Secure Internet Communications from Man-in-the-Middle Attacks? In *In proceedings of CNSM:International Conference on Network and Service Management*, 2017.
[39] H.-D.-D. Nguyen, C.-D. Phung, S. Secci, B. Felix, and M. Nogueira. Can MPTCP Secure Internet Communications from Man-in-the-Middle Attacks? 2017.
[40] V. Olteanu, A. Agache, A. Voinescu, and C. Raiciu. Stateless datacenter load-balancing with beamer. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 125–139, Renton, WA, 2018. USENIX Association.
[41] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D. A. Maltz, R. K̃ern, H. Kumar, M. Zikos, H. Wu, C. Kim, and N. Karri. Ananta: Cloud scale load balancing. In *SIGCOMM*, 2013.
[42] Perrin T. Axolotl ratchet. https://github.com/trevp/axolotl/wiki.
[43] C. Raiciu, D. Niculescu, M. Bagnulo, and M. J. Handley. Opportunistic mobility with multipath tcp. In *Proceedings of the Sixth International Workshop on MobiArch*, MobiArch '11, pages 7–12, New York, NY, USA, 2011. ACM.
[44] E. Rescorla. The transport layer security (TLS) protocol version 1.3. internet draft. 2018.
[45] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, Nov. 1979.
[46] Signal. Signal documentation. https://signal.org/docs/.
[47] Y. Takano, N. Isozaki, and Y. Shinoda. Multipath key exchange on p2p networks. In *First International Conference on Availability, Reliability and Security (ARES'06)*, April 2006.
[48] The Intercept. The NSA's Spy Hub in New York, Hidden in Plain Sight. https://theintercept.com/2016/11/16/the-nsas-spy-hub-in-new-york-hidden-in-plain-sight/. Last accessed: November 2017.
[49] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith. Sok: secure messaging. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 232–249. IEEE, 2015.
[50] D. Wendlandt, D. G. Andersen, and A. Perrig. Perspectives: Improving ssh-style host authentication with multi-path probing. In *USENIX Annual Technical Conference*, volume 8, pages 321–334, 2008.
[51] Wired. WHAT WE KNOW ABOUT THE NSA AND AT&T'S SPYING PACT. https://www.wired.com/2015/08/know-nsa-atts-spying-pact/. Last accessed: November 2017.
[52] F. L. Wong and F. Stajano. Multichannel security protocols. *Pervasive Computing, IEEE*, 6(4):31–39, 2007.

A DETAILED SECURITY MODEL

To analyze SMKEX, we extend the model of Canetti and Krawczyk [6] in several aspects and describe how the protocol is analyzed in that framework. We first shortly recall the Canetti-Krawczyk model in §A.1, and then describe the necessary modifications in §A.2.

A.1 Canetti-Krawczyk in a nutshell

Message-driven protocols. The model of Canetti and Krawczyk [6] models the execution of a protocol, modeled through a Turing machine, between a set of parties $P_1, \ldots, P_n$. A *message-driven protocol* is either triggered at a party through an external "call" or through an arriving messages. At each of these events, the protocol processes the incoming information and may produce local output and/or transmit a message over the network. Local outputs are explicitly labeled as either "public" or "secret". Typically, a message driven protocol is initiated through an external call at one party and then proceeds by sending messages back and forth until the protocol is finished.

Key-exchange protocols. A *key-exchange protocol* is a message-driven protocol in which the interaction proceeds between pairs of parties and which return, upon completion, a secret key called a *session key*. The protocol is initiated at a party P_i through an external call (P_i, P_i, *sid*, *role*), where P_j is the identity of the intended partner and *role* is either initiate or respond, depending on whether the party sends the first message.

Session-key security. Security of a key-exchange protocol is defined through a game in which an adversary $\mathcal{A}$ performs queries to certain oracles given to it. In more detail, these oracles are as follows:

- New session. Adversary $\mathcal{A}$ can create a new key-exchange session at a party P_i via a query (P_i, P_i, *sid*, *role*), and obtains the potential public outputs and messages generated by the protocol invocation.
- Deliver message. Adversary $\mathcal{A}$ can deliver a message m to a party P_j, which results in the invocation of the protocol given message m. The adversary again obtains the potential public outputs and messages generated by the protocol invocation. The fact that the adversary can deliver arbitrary messages models that it is assumed to completely control the network.
- Session-state reveal. Adversary $\mathcal{A}$ specifies a party P_i and a protocol session *sid* and obtains the state of an active but incomplete protocol session. The exact information that is leaked through this query is specified by each protocol.
- Session-key query. Adversary $\mathcal{A}$ specifies a party P_i and a protocol session *sid* and obtains all "secret" outputs of the session.
- Party corruption. Adversary $\mathcal{A}$ specifies a party P_i and obtains the long-term secrets of P_i. (This oracle call is actually not needed for SMKEX.)
- Session expiration. Adversary $\mathcal{A}$ specifies a party P_i and a protocol session *sid*. This deletes all data generated of that session, a feature that is necessary for modeling forward secrecy of key-exchange protocols.
- Session test. Adversary $\mathcal{A}$ specifies a party P_i and a protocol session *sid*. Depending on a hidden, uniformly random bit $b \in \{0, 1\}$, adversary $\mathcal{A}$ obtains (i) either a uniformly random session key or (ii) the actual key computed by P_i in session *sid*.

The choice of test session is, however, restricted; adversary $\mathcal{A}$ can only choose sessions at (i) uncorrupted parties, where (ii) the session key has not been queried and (iii) the session state has not been revealed. The same restrictions also apply to the partner session; i.e. the session at some party P_j that shares the same session identifier *sid*. To model forward secrecy, the choice of expired sessions at corrupted parties is also allowed.

Finally, adversary $\mathcal{A}$ outputs a bit b' which can be understood as a guess for b, and $\mathcal{A}$ is said to win the game if $b = b'$ or if there are mismatching keys in some session. The advantage of $\mathcal{A}$ is then defined as $\mathrm{Adv}_{\mathrm{SK}}(\mathcal{A}) = 2\Pr[\mathcal{A} \text{ wins}] - 1$.

Often, the security definition is understood in an asymptotic sense; however, we prefer a concrete-security treatment explicitly specifying the advantages to obtain more useful guarantees for real-world use.

A.2 Formal model of protocol and sessions

We first observe that SMKEX can be seen as a message-driven protocol as specified in Section A.1. We slightly modify the network model to incorporate differentiation between the two network endpoints associated to each initiator by addressing all initiators P by either $P.1$ or $P.2$, depending on whether the message is sent via the first or the second channel, and the initiator protocol signals to the adversary whether a message appears as sent from $P.1$ or $P.2$. Likewise, when a initiator protocol is invoked by the adversary with a message from the network, it is signaled whether the message is received on the first or the second channel. The initiator P_i, as in the definition of key-exchange protocol in [6], initially obtains an invocation (P_i, P_j, initiate, *id*). The initiator uses the invocation to send a message to the responder, which correspond to the message A1 → B1 in the diagram. Upon a second invocation (P_i, P_j, follow-up, *id*), the initiator sends a second (possibly empty) message to the responder, which corresponds to the message A2 → B2 in the diagram. The responder need not receive such an explicit invocation, as the sessions are started by the messages received on the network. The value *id* is only used locally by P_i to identify *which* one of the possibly multiple sessions between P_i and P_j is referred to. The responder answers both messages according to the SMKEX message flow (i.e., Ai → Bi is answered by Bi → Ai), and outputs the secret

key and the session ID *sid*, which is generated by the protocol and different from *id*. The initiator, after receiving both messages, also computes the key.

The formal specification of protocol session then follows more closely the variant of Choo et al. [9], in which the session ID of a protocol is considered as an output—not an input—of the protocol. It is specified as the concatenation of all messages sent and received by a protocol.

Rationale. Canetti and Krawczyk [6] consider the session ID as an input to the protocol. In the opportunistic-encryption setting, however, it seems unnatural to assume that the initiator and responder already share a session ID prior to the start of the session. Therefore, we adapt a modification of Choo et al. [9] in which the session ID is defined as the concatenation of all protocol messages; this can be seen as an analogue to the model of Bellare and Rogaway [3]. We also drop the requirement of the responder's protocol being invoked by an explicit message (P_j, P_i, respond); the corresponding input is needed in [6] to define the session. The second, possibly empty, message A2 → B2 sent by the initiator is included for definitional reasons. Depending on the exact definition we consider, the two messages sent by the responder may have to be given to different instances of the adversary—therefore they must not be returned in the same oracle call. Generating the message B1 → A1 as a response to the actual message and the message B2 → A2 as a response to a subsequent empty message is one way of modeling the protocol appropriately for this.

A.3 *A-P* and *A/A* adversaries

We first observe that all considered adversaries can emulate a client to a server; therefore, the fact that a server session completes and accepts the key although it does not communicate with the actual client is not considered an attack. This resembles the server-only authentication scenario considered in previous work such as by Krawczyk et al. [28], and analogously with that work we require that the test session chosen by the adversary must be a client (or: initiator) session.

A-P adversary. We require for the test session that the adversary $\mathcal{A}$ delivers at least one of the messages B1 → A1 or B2 → A2 unmodified to the client.

A/A adversary. This setting is a bit more complicated, because we must have two active adversaries which are not allowed to communicate. Each active adversary must obtain one message (B1 → A1 resp. B2 → A2) and deliver it to the client. In between, we may want to allow the adversaries to query other oracles, but they must not be able to use this for covert communication.

We formalize this as follows. At any point during the game, the adversary $\mathcal{A}$ can output two states $S, T \in \{0,1\}^*$. The game then proceeds in the following steps:

(1) Invoke $\mathcal{A}$ with input S to obtain output S', this stage of the adversary is also denoted as $\mathcal{A}_1$,
(2) invoke $\mathcal{A}$ with input T to obtain output T', this stage of the adversary is also denoted as $\mathcal{A}_2$,
(3) invoke $\mathcal{A}$ with input S' to obtain output S'', this is again $\mathcal{A}_1$,
(4) invoke $\mathcal{A}$ with input T' to obtain output T'', this is again $\mathcal{A}_2$,
(5) finally, invoke $\mathcal{A}$ with input (S'', T'') and continue the experiment as before.

By the definition of an A/A adversary, $\mathcal{A}_1$ and $\mathcal{A}_2$ must not communicate during the attack. This is formalized in our model by the following conditions:

- the query (P_i, P_j, initiate, *id*) to P_i in session *sid*, as well as the queries to deliver the messages A1 → B1 to P_j (i.e., messages from $P_i.1$) and B1 → A1 to P_i (i.e., to $P_i.1$) in session *sid* were made during steps 1 or 3, i.e. by $\mathcal{A}_1$,
- the query (P_i, P_j, follow-up, *id*) to P_i in session *sid*, as well as the queries to deliver the messages A2 → B2 to P_j and B2 → A2 to P_i in session *sid* were made during steps 2 or 4, i.e. by $\mathcal{A}_2$,
- adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ interact (in the sense of activating, revealing state, or corrupting) with disjoint sets of parties; that is, there is no party P_n such that both $\mathcal{A}_1$ and $\mathcal{A}_2$ make a query targeting P_n.

Furthermore, the freshness requirements of the original definition still apply [6].

The model described above is valid only for one-round protocols, as the number of activations admitted to each adversary allows to deliver only one message on each of the four channels A1 → B1, A2 → B2, B1 → A1, and B2 → A2. For multi-round protocols, the model can be extended by invoking $\mathcal{A}_1$ and $\mathcal{A}_2$ in an alternating order for multiple rounds. For one-round protocols, the two definitions are equivalent, as the views of $\mathcal{A}_1$ and $\mathcal{A}_2$, conditioned on the history up to the point where they were split, are statistically independent, and therefore spreading their execution over multiple phases does not give them an advantage.

Rationale. The idea behind splitting the adversary $\mathcal{A}$ during the interaction with the test session into two invocations is to model the fact that, with an A/A adversary, the two attackers controlling the links A1 ↔ B1 and A2 ↔ B2 cannot communicate. Before the SMKEX test session is initiated at the sender, both attackers are allowed to interact, which is modeled by having $\mathcal{A}$ output S, T during the game. During the session, the attackers may not interact—this is achieved by running $\mathcal{A}$ on inputs S and T independently, by requiring that $\mathcal{A}$ only obtain information about one link A1 ↔ B1 or A2 ↔ B2, respectively, and by restricting access to the other oracles to prevent covert communication. The conditions stated above faithfully model this isolation requirement of an A/A adversary, while the definition at the same time allows $\mathcal{A}$ to choose the test session adaptively and after the execution of the session is finished. Overall, the definition provides maximum flexibility to $\mathcal{A}$ under the condition of being an A/A adversary.

A.4 Further preliminaries

The security of SMKEX depends on the hardness of the Decisional Diffie-Hellman (DDH) problem in a given group $\mathbb{G}$. The advantage of an adversary in solving the DDH problem is defined as follows.

Definition 3. *Let $\mathcal{A}$ be an algorithm, and $\mathbb{G}$ be a cyclic group with generator g. The* advantage of $\mathcal{A}$ in solving the DDH problem in $\mathbb{G} = \langle g \rangle$ *is defined as*

$$\mathrm{Adv}^{\mathbb{G}}_{DDH}(\mathcal{A}) := \Pr[\mathcal{A}(g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(g^a, g^b, g^c) = 1]\,,$$

where $a, b, c \in \{1, \ldots, \#\mathbb{G}\}$ are chosen uniformly at random.

B SECURITY PROOFS

B.1 Protocol specification

The protocol generates uniformly random keys from the group $\mathbb{G}$ and works as follows, for Diffie-Hellman in group $\mathbb{G} = \langle g \rangle$ with order q:

- (Initiator) On input (P_i, P_j, initiate, *id*), choose a random exponent $x \in \mathbb{Z}_q$. Send (id, g^x), intended as message A1 → B1 to P_j; i.e., the message is sent with sender address $P_i.1$.
- (Initiator) On input (P_i, P_j, follow-up, *id*), choose a random nonce $N_C \in \{0,1\}^\nu$ and send message (id, N_C), intended as message A2 → B2, to P_j; i.e., the message is sent with sender address $P_i.2$.
- (Responder) Receiving the first message (id, g^x) from $P_i.1$, choose a random nonce $N_S \in \{0,1\}^\nu$ and an exponent $y \in \mathbb{Z}_q$. Respond with (id, g^y) to $P_i.1$; this is intended as message B1 → A1.
- (Responder) Receiving the message (id, N_C) from $P_i.2$, compute $h = H(N_C, g^x, N_S, g^y)$ and send the response (id, N_S, h) to $P_i.2$, intended as message B2 → A2. Also, compute the session id *sid* ←

(N_C, g^x, N_S, g^y, h) and the secret key $k \leftarrow g^{xy}$ and output locally (P_i, P_i, sid, k).

- (Initiator) Receiving the first response message (id, g^y) from P_j in a given session, store it in the session state.
- (Initiator) Receiving a message (id, N_S, h') from P_j in a given session, check whether $h' = H(N_C, g^x, N_S, g^y)$. Abort if the check fails. Otherwise, compute *sid* and k analogously to the responder and output $\left(P_i, P_j, sid, k\right)$ locally.

B.2 Analysis in the A-P model

The proofs in this section could also be given in the standard model, assuming collision resistance of the hash function. If the first path is controlled by a passive attacker, the hash function is not needed. If the second path is controlled by a passive attacker, then collision resistance is sufficient to prevent the attacker from modifying the messages on the first path. Yet, as the proof in §B.3 crucially depends on the random-oracle model, we decided to provide the proofs in this section in the same model.

Model the hash function as a random oracle with output space $\{0,1\}^\lambda$. Theorem 1 follows as a combination of Lemmas 4 and 5, by choosing either of the adversaries $\mathcal{B}$ described in the lemmas at random.

Lemma 4. *Let $\mathcal{A}$ be an adversary that makes at most q queries to the random oracle and initiates at most s sessions. $\mathcal{A}$ behaves as an A-P adversary in the test session and keeps the message* B2 → A2 *constant. Then there is an adversary $\mathcal{B}$, described in the proof, such that*

$$\mathrm{Adv}_{sK}(\mathcal{A}) \leq s\mathrm{Adv}^{\mathbb{G}}_{DDH}(\mathcal{B}) + sq/2^\lambda \ .$$

Proof. The proof proceeds via a sequence of games, starting with G_0, which is the game specified in Section A. In all games, we denote by g^x the message sent as A1 → B1 in the test session, and by N_C the message sent as A2 → B2. We denote by g^y and (N_S, h) the messages B1 → A1 and B2 → A2 sent in the test session, respectively. That is, we consider the session $sid = (N_C, g^x, N_S, g^y, h)$. Note that, at this point, the test session and its transcript are defined via the adversary's test query P_i, and there may not be a corresponding session at P_j that has a matching transcript. We consider adversaries that choose their test sessions non-adaptively; a straightforward reduction choosing a uniformly random one out of the up to s sessions shows that for such an $\mathcal{A}'$ it holds that $\mathrm{Adv}_{\mathrm{SK}}(\mathcal{A}) \leq s\mathrm{Adv}_{\mathrm{SK}}(\mathcal{A}')$.

The next game G_1 is defined as G_0, but the adversary loses if it queries the random oracle on some input $X \neq (N_C, g^x, N_S, g^y)$, and this results in output h. By the Union Bound, the probability of this event occurring is bounded by $q/2^\lambda$. That means $\Pr[G_0] \leq \Pr[G_1] + q/2^\lambda$.

We now observe that, if $\mathcal{A}$ wins in G_1, then there is always a matching session for the test session at P_j. By assumption, the value h is delivered unmodified to P_i, from some session at P_j. As no collision in the random oracle occurs in G_1 and P_i accepts, the transcript of P_j in that session must be exactly the same as that of P_i; the session also computes the same identifier *sid*. Furthermore, as the key is fully determined by the transcript, as well as by the correctness of the protocol and the fact that the transcripts match, the keys computed by P_i and P_j in session *sid* are equal.

We finally describe an adversary $\mathcal{B}$ that plays the DDH game and emulates G_1 to $\mathcal{A}$, embedding the DDH challenge in the test session. This adversary $\mathcal{B}$ simulates game G_1 to $\mathcal{A}$ by emulating all oracles. In the test session, $\mathcal{B}$ computes the messages almost as described by the protocol, but embeds the Diffie-Hellman triple (g^a, g^b, g^c) obtained from the DDH game by using g^a in the message A1 → B1, g^b in the message B1 → A1, and g^c as the key revealed through the test query. All other sessions are simulated independently of the test session using fresh randomness; in particular, corruption, key and state reveal queries can be emulated easily in those sessions. On the other hand, such queries are disallowed if they affect the test session. Overall, $\mathcal{B}$ provides $\mathcal{A}$ with a perfect emulation of the game (as by the above assumption, the test session is known). This concludes the proof. □

Lemma 5. *Let $\mathcal{A}$ be an adversary that initiates at most s sessions. $\mathcal{A}$ behaves as an A-P adversary in the test session, and keeps the message* B1 → A1 *constant. Then there is an adversary $\mathcal{B}$, described in the proof, such that*

$$\mathrm{Adv}_{sK}(\mathcal{A}) \leq s\mathrm{Adv}^{\mathbb{G}}_{DDH}(\mathcal{B}) \ .$$

Proof. The proof proceeds as the one of Lemma 4, but we can even spare the game hop. For simplicity we denote the game as G, and use the same notation for the messages as above.

If $\mathcal{A}$ wins in G, then there is always a matching session for the test session at P_j. By assumption, the messages A1 → B1 and B1 → A1 are delivered unmodified between P_i and P_j. As P_i accepts, the message h received by P_i in that session must match the one sent by P_j; the session also computes the same identifier *sid*. As in Lemma 4, the keys computed by P_i and P_j in session *sid* are equal.

The adversary $\mathcal{B}$ that plays the DDH game and emulates G to $\mathcal{A}$ is described analogously to the one in Lemma 4. □

B.3 Proof of analysis in the A/A model

The proof proceeds in a sequence of games, starting with G_0, which is the game specified in Section A for the A/A adversary. Variable naming is as in Lemma 4, and we also start by guessing the test session also as in that lemma.

The next game G_1 is defined as G_0, but the adversary loses if the nonce N_C chosen by P_i in the test session is consistent with any prior random-oracle query by $\mathcal{A}$, i.e., if there has been a query $(N_C, \cdot, N_S, \cdot)$ or $(\cdot, g^x, \cdot, \cdot)$ to the random oracle. As N_C, N_S, and g^x are chosen uniformly at random, the probabilities of these events are $q/2^{\nu-1}$ and $q/\#\mathbb{G}$, respectively, and $\Pr[G_0] \leq s\Pr[G_1] + sq/2^{\nu-1} + sq/\#\mathbb{G}$.

The next game G_2 is defined as G_1, with two differences. First, the challenger guesses the session at P_j that will be partnered with the test session. Note that the adversary can forward the initial message from the test session to P_j repeatedly and thereby start multiple sessions at the side of P_j. In the reduction, however, we need to adapt the random oracle in the correct session; therefore, we have to guess which one of the at most t sessions at P_j will be completed. Furthermore, a value $\tilde{h} \leftarrow\!\!\$\ \{0,1\}^\lambda$ is chosen in the beginning of the game. This value $\tilde{h}$ is then used as the output of the random oracle in the session corresponding to the test session at P_j. Overall, this means that $\Pr[G_1] \leq 1/t \cdot \Pr[G_2]$.

The next game G_3 is defined as G_2, but the random oracle given to $\mathcal{A}_2$ is redefined to sample uniform random values for all inputs $(\cdot, g^x, \cdot, \cdot)$, which are independent of the random oracle used in the remainder of the game. Furthermore, $\mathcal{A}$ loses in case any such query occurs. Note that the model of protocol execution for valid A/A adversaries, together with pre-sampling the value $\tilde{h}$, ensures that the execution of $\mathcal{A}_2$ is independent of the execution of $\mathcal{A}_1$, given the execution of $\mathcal{A}$ up to the point where it splits and the value $\tilde{h}$. As x is chosen independently of the previous state within the execution of $\mathcal{A}_1$, and by the Union bound, we observe that the probability for $\mathcal{A}_2$ making such a query to the random oracle is bounded by $q/(2^\nu \#\mathbb{G})$. It follows that $\Pr[G_2] \leq \Pr[G_3] + q/\#\mathbb{G}$.

In the next game G_4, a similar modification is made with respect to the random oracle given to $\mathcal{A}_1$ on inputs $(N_C, \cdot, N_S, \cdot)$. By the analogous arguments, $\Pr[G_3] \leq \Pr[G_4] + q/2^{2\nu}$.

In the next game G_5, the adversary also loses if there is no session at P_j that matches the test session at P_i. The fact that P_i accepts means that the check value $h' \leftarrow H(N_C, g^x, N_S, g^y)$ computed by P_i matches the value h generated by $\mathcal{A}_2$ and delivered to P_i as message B2 → A2. As, for any value g^y other than those generated by P_j, the value h' is chosen independently of the complete execution of $\mathcal{A}_2$ that generates h, the probability of this

event to occur (i.e., that the test session has no matching session at P_i) is bounded by $1/2^\lambda$. Therefore, $\Pr[G_4] \leq \Pr[G_5] + 1/2^\lambda$.

As the client and server sessions match, we can again reduce to DDH via an adversary $\mathcal{B}$ that embeds the DDH challenge into the test session and simulates the remaining sessions; this step is analogous to the proofs of the above lemmas. Finally, the theorem follows from combining all the above equations.

C VARIANTS OF THE SMKEX PROTOCOL

Here we describe several variants of the SMKEX protocol and discuss their respective security features.

C.1 Sending an additional hash to the server

The protocol can be modified to send a hash $h = H(N_C, g^x, N_S, g^y, 0)$ from P_j to P_i and another hash $\bar{h} = H(N_C, g^x, N_S, g^y, 1)$ from P_i to P_j. (This can alternatively be implemented by sending the first half of a hash value computed as before from P_j to P_i and the second half from P_i to P_j; modulo adapting the error bound this is equivalent in the random-oracle model.)

The advantage of this modification is that the server verifies that the client controls both endpoints A1 and A2. While this may be advantageous in certain scenarios, we do not consider it helpful in the setting considered in this paper.

More formally, this additional guarantee is reflected in the security guarantee by allowing $\mathcal{A}$ to additionally choose sessions at the responder P_j as the test session. With this strengthened definition, we can then prove the following theorem analogously to the main result.

Theorem 6. *Let $\mathcal{A}$ be an A/A adversary that makes at most q queries to the random oracle, initiates at most s sessions at clients and at most t sessions at servers. Then there is an adversary $\mathcal{B}$, described in the proof, such that*

$$\mathrm{Adv}_{SK}(\mathcal{A}) \leq 2st\,\mathrm{Adv}_{DDH}(\mathcal{B}) + stq/(2^{\nu-2}\#\mathbb{G}) + st/2^{\lambda-1}\ .$$

Proof. The proof follows overall along the same lines as that of Theorem 2. The main difference is, however, that initially we insert a reduction step that guesses whether the test session will be at the *initiator* or *responder*, which increases the overall reduction slack by a factor of 2. In case the test session is at the *initiator*, the proof continues exactly as Theorem 2.

In case the test session is at the *responder*, the pre-sampled value $\tilde{h}$ is used in the second message A2 → B2. The argument for the correct distribution of $\tilde{h}$ based on the unguessability of N_C proceeds exactly as in Theorem 2. The argument for the existence of a matching session also proceeds similarly, albeit with exchanged roles for P_i and P_j. This concludes the proof. □

C.2 SMKEX security for n paths

Expanding to more paths is desirable. If the paths are disjoint, this increases the probability that at least one path lacks an active attacker or has an attacker that cannot synchronize with the others.

Assume the two communicating parties are connected via n paths. In this case, we have n attackers, each possibly active or passive, and $n(n-1)/2$ relationships between attackers. While tackling every combination is unfeasible, we show how our SMKEX protocol can be generalized to reduce the n paths case to a 2 paths case.

Let $\alpha_i \to \beta_i$ and $\beta_i \to \alpha_i$, for $i = 1..n$, be the messages exchanged across the n paths between the two parties wanting to perform a key exchange via n-paths SMKEX. We define the attacker for each pair of messages $\alpha_i \to \beta_i$ and $\beta_i \to \alpha_i$ as $\mathcal{B}_i$.

Without loss of generality, we assume the messages $\alpha_1 \to \beta_1$ and $\beta_1 \to \alpha_1$ are identical to the $A_1 \to B_1$ and $B_1 \to A_1$ messages, respectively, in the normal 2-path version of SMKEX. Additionally, for all $i = 2..n$, $\alpha_i \to \beta_i$ and $\beta_i \to \alpha_i$ are identical to $A_2 \to B_2$ and $B_2 \to A_2$, respectively.

We build an undirected graph with the attackers as vertices. We then add an edge to the graph for each pair of attackers that are *synchronized*.

If the graph is connected and all the attackers are active, we are in the A-A case and security does not hold. We now prove that security holds when at least one of the following is true: the graph is not connected (which we reduce to the A/A case), or at least one of the attackers is passive (which we reduce to the A-P case).

If the graph is not connected, we split the attackers into two sets S and S' such that no edge exists between vertices in different sets. We view all the attackers in S as a single active attacker $\mathcal{A}$, and all the attackers in S as a second active attacker $\mathcal{A}'$. Since no edges exist between S and S', it implies that $\mathcal{A}$ and $\mathcal{A}'$ are *unsynchronized*. It quickly follows that (in the worst case) this is the A/A case for 2 paths. Note that in this case some of the hashes might be accepted (for example, if there are synchronized active attackers in the set that sees all 4 messages); thus, the protocol must accept the key only if all the hashes are accepted.

If at least one of the attackers is passive, we take all the passive attackers (of which there is at least one) and put them in a set S. We then put the rest of the attackers in set S'. We view all the attackers in S as a single passive attacker $\mathcal{A}$, and all the attackers in S' as a single active attacker $\mathcal{A}'$. It quickly follows that, in the worst case, this is the A-P case for 2 paths. Just as before, the protocol must accept the key only if all the hashes are accepted.

C.3 Double ratcheting

Double ratcheting has been used by Axolotl [42] and is currently used by its popular successor, Signal [46], in order to provide forward and backward secrecy (aka future secrecy) [11, 49].

While our SMKEX protocol provides forward and backward secrecy across sessions by default (since we use independent ephemeral keys per session), it might be useful to provide forward and backward secrecy also across messages from a single session. In this case, we can implement the double ratcheting protocol from Signal [11, 46]: a) (asymmetric ratcheting) by sending public keys with each message and refreshing these keys every few messages and then updating the symmetric encryption keys based on these newly exchanged public keys; b) (symmetric ratcheting) by deriving new message encryption keys for each message and deleting the old ones.

The disadvantage of this approach is that clients and servers need to maintain a larger key state and for messages received out of order we need to keep old keys in memory, which goes against forward secrecy. Hence, this solution might be useful for applications using long-lived sessions, but the overhead might not be worth for short-lived sessions.

D STANDARD-MODEL SOLUTION

In this section we describe a protocol that does not require the use of the random-oracle model in the proof of security. This protocol uses strongly non-malleable codes for encoding the messages sent from the server to the client. This section contains the fundamental material as well as the protocol and the analysis.

D.1 Preliminaries for non-malleable codes

On a high level, a non-malleable code is an encoding scheme which guarantees that, for a certain class of tampering functions, decoding a tampered codeword either leads to the correct message or to a message that is independent of that correct message [16]. In that sense, non-malleable codes are weaker than error-correcting codes.

Definition 4. *A (k, n)-coding scheme* (Enc, Dec) *consists of a randomized encoding function* $\mathrm{Enc} : \{0,1\}^k \to \{0,1\}^n$ *and a deterministic decoding function* $\mathrm{Dec} : \{0,1\}^n \to \{0,1\}^k \cup \{\perp\}$ *such that* $\mathrm{Dec}(\mathrm{Enc}(x)) = x$ *(with probability* 1 *over the randomness of the encoding function) for each $x \in \{0,1\}^k$. The special symbol $\perp$ indicates an* invalid *codeword.*

In particular, for a split-state non-malleable code there are $k_1 + k_2 = k$ such that independently modifying the first k_1 and the last k_2 bits will yield either the original plaintext, or an independently sampled one, or $\perp$.

Definition 5 (Strong non-malleability). *Let $\mathcal{F}$ be a family of functions. A (k, n)-coding scheme* (Enc, Dec) *is ε-*strongly non-malleable *w.r.t. $\mathcal{F}$ and adversary $\mathcal{A}$ if for any $s_0, s_1 \in \{0,1\}^k$ and any $f \in \mathcal{F}$, we have*

$$\mathrm{Adv}^{f,s_0,s_1}_{\text{S-NMC}}(\mathcal{A}) = \Pr\left[\mathcal{A}\left(\mathsf{StrongNM}^f_{s_0}\right)\right] - \Pr\left[\mathcal{A}\left(\mathsf{StrongNM}^f_{s_1}\right)\right]$$

where

$$\mathsf{StrongNM}^f_s := \left\{ \begin{array}{c} c \leftarrow \mathrm{Enc}(s); \tilde{c} \leftarrow f(c); \tilde{s} \leftarrow \mathrm{Dec}(\tilde{c}) \\ \textit{Output } \mathsf{same}^* \textit{ if } c = \tilde{c}\textit{, and } \tilde{s} \textit{ otherwise} \end{array} \right\} .$$

The value same^* in Definition 5 is a constant that is different from any value that can be encoded by the non-malleable code. The reason is that a function f can leave the codeword unmodified, and so the two distributions in the definition of the advantage become clearly distinguishable, as one outputs s_0 while the other one outputs s_1. Replacing these values by the same constant same^* is both necessary and sufficient to make the definition achievable.

D.2 The protocol

The protocol works similarly to SMKEX in the random oracle model. In more detail:

- (Initiator) The initiator sends g^x as A1 $\rightarrow$ B1 and N_C as A2 $\rightarrow$ B2, for the same choice of values as before.
- (Responder) Encode $m \leftarrow \langle (N_C, g^x, N_S, g^y) \rangle$ as a bit string, and use the non-malleable code to obtain $c \leftarrow \mathrm{Enc}(m)$. Then, set $c_1, c_2 \leftarrow c$ with $|c_1| = k_1$ and $|c_2| = k_2$. Send c_1 as B1 $\rightarrow$ A1 and c_2 as B2 $\rightarrow$ A2. Compute and output the key as usual.
- (Initiator) Receive c_1 and c_2, set $m' \leftarrow \mathrm{Dec}(c_1 \| c_2)$. If $m' = \perp$ then abort, if m' does not start with N_C then abort. Else, decode m' to obtain g^y and N_S and proceed as usual.

The protocol can be seen as an abstraction of SMKEX; indeed, SMKEX can be viewed as the above protocol with a specific non-malleable code based on hash functions. Yet, formalizing the protocol in terms of non-malleable codes allows us to build a protocol secure in the standard model, by usiing (for instance) the results in [2].

D.3 The security analysis for A/A adversaries

The security proof follows along the same line as in Theorem 2, but instead of programming the random oracle we have to reduce from the security of the strongly non-malleable code.

Theorem 7. *Let $\mathcal{A}$ be an A/A adversary that initiates at most s sessions at clients and at most t sessions at servers. Then there is are adversaries $\mathcal{B}$, C_1, C_2, described in the proof, such that*

$$\mathrm{Adv}_{SK}(\mathcal{A}) \leq st\mathrm{Adv}^{\mathbb{G}}_{DDH}(\mathcal{B}) + st/2^{\nu} + st\mathrm{Adv}_{\text{S-NMC}}(C_1) + st\mathrm{Adv}_{\text{S-NMC}}(C_2) .$$

Proof. The first step as above is that we guess the session that will be chosen as a test session, losing a factor st. Otherwise, the first game is the actual SK-security game.

The next game G_1 idealizes the strongly non-malleable code. That is, in the test session, when the concatenation $c'_1 \| c'_2$ of the two messages c'_1 and c_2 provided by $\mathcal{A}_1$ and $\mathcal{A}_2$ in the test session does not exactly match the concatenation $c_1 \| c_2$ of messages sent by P_j, then the test session at P_i aborts and the adversary loses. We now show that $\Pr[\mathrm{G}_1] \leq \Pr[\mathrm{G}_2] + 2^{-\nu} + \mathrm{Adv}^{f,s_0,s_1}_{\text{NMC}}(C_1) + \mathrm{Adv}^{f,s_0,s_1}_{\text{NMC}}(C_2)$ for distinguishers C_1 and C_2, split-state function f and input values s_0, s_1 as described below.

The SK-security game can be changed such that the values N_C, N_S, x, y used in the test session are pre-sampled. Message s_0 is then defined as (N_C, g^x, N_S), and message s_1 is the all-0 message of the same length. Function f is defined through emulating the SK-security game to $\mathcal{A}$, with the values N_C, N_S, x, y as above. This function f is a split-state function, as the first k_1 bits of the message are processed through $\mathcal{A}_1$ and the following k_2 bits are processed through $\mathcal{A}_2$, and there is no interaction between the two by the definition of a valid A/A adversary $\mathcal{A}$. The distinguisher C_1 is then defined as outputting 1 if the message is exactly (N_C, g^x, N_S, g^y) and 0 otherwise. Distinguisher C_2 outputs 1 if the message is $(N_C, \cdot, \cdot, \cdot)$ and 0 otherwise.

The argument then proceeds as follows: If the test session aborts, the probability of winning in both G_1 and G_2 is exactly $1/2$, since everything else is independent of it. Also, if the function f keeps the value $c_1 \| c_2$ constant is the same in both cases, since by the correctness of the code, the games lead to the same result. Therefore, there are two differences that can be exploited are the probability of getting a message encoding $(N_C, \cdot, \cdot, \cdot)$ which is not the same as the original one (this leads to the terms $\mathrm{Adv}^{f,s_0,s_1}_{\text{S-NMC}}(C_2)$ and $2^{-\nu}$) and a different probability in leading to exactly (N_C, g^x, N_S, g^y) (this leads to the term $\mathrm{Adv}^{f,s_0,s_1}_{\text{S-NMC}}(C_2)$.

The remainder of the proof proceeds just as in Theorem 2. □

D.4 Strengthening the code for A-P adversaries

For arbitrary strongly non-malleable codes, the construction is not necessarily secure against A-P adversaries. The reason is that the split-state functions requires the two parts of the functions to tamper with the codeword parts independents; an A-P adversary corresponds to tampering with only one part of the codeword, but instead the second part to the codeword is known to this function.

Recall the notion of a commitment scheme (Com, Open) where Com is a probabilistic algorithm that takes as input a message x and produces as output a commitment c and opening information r. Algorithm Open takes as input a message x', commitment c, and opening information r', and outputs a bit that indicates whether or not the message contained in c is actually $x' = x$.

Commitments are usually required to be *hiding* in the sense that c does not leak information about the contained message x, and *binding* in the sense that it is (at least computationally) difficult to produce an opening information r' that makes Open accept a message $x' \neq x$.

A strongly non-malleable code can be modified to also work against such types of attacks by (for instance) including a hash of a codeword part in the respective other part of the codeword. More formally, for a codeword $e = e_1 | e_2$ of a split-state non-malleable code, compute commitments $(c_1, r_1) \leftarrow\!\!\$\ \mathrm{Com}(e_1)$ and $(c_2, r_2) \leftarrow\!\!\$\ \mathrm{Com}(e_2)$ and set the messages to $m_1 \leftarrow e_1 | r_1 | c_2$ and $m_2 \leftarrow e_2 | r_2 | c_1$.

Using an information-theoretically hiding commitment means that an A/A adversary does not gain a noticeable advantage by the additional information; for instance, in $m_1 = e_1 | r_1 | c_2$ the opening information r_1 is merely an independent random string and the commitment c_2 does not contain significant information on e_2; therefore, mauling c_1 has not become significantly easier than for a plain code.

For an A/P adversary, however, either one of the messages m_1 and m_2 is transmitted unmodified, say m_1. In that case e_1 is transmitted to A1 without modification, as is c_2 which then, by the binding property of the commitment, also ensures that e_2 is transmitted unmodified.