SEKA: Secretless Key Exchange and Authentication in LiFi Networks

Eric Ackermann
CISPA[§]
eric.ackermann@cispa.de

Kai Lennert Bober, Volker Jungnickel
Fraunhofer HHI
{*kai.lennert.bober,volker.jungnickel*}*@hhi.fraunhofer.de*

Anja Lehmann
Hasso-Plattner-Institute,
University of Potsdam
anja.lehmann@hpi.de

***Abstract*—Light Fidelity (LiFi) networks transmit information via light waves and are an interesting alternative to Radio Frequency networks: as light can be confined easily, LiFi provides better performance and makes eavesdropping attacks much more difficult. A core application of LiFi networks is self-contained and local networks among a group of autonomous devices, e.g., in industrial or medical environments. Cryptographic protocols are used to secure these networks, however the key exchange sometimes relies solely on the confineability of light signals and sends key material in plain over the network. This is clearly not desirable from a security perspective and newer standards recommend key exchange protocols to establish shared keys. A crucial part in any authenticated key exchange protocol is how to bootstrap trust, e.g., by assuming a PKI, pre-installed keys or an out-of-band-channel. Well established solutions exist, but they are not ideal for the type of self-contained networks targeted by LiFi communication.**

In this work we investigate how the physical properties of a LiFi channel can be used to replace these mechanisms, resulting in a more convenient and also more efficient solution for key exchange. To this end we propose a new type of *secret-less key exchange* (SEKA) that does not rely on any pre-shared secrets, and instead runs in two phases: a short bootstrap phase where we make stronger assumptions on the physical security, ruling out active attacks. This can be realized by putting all devices in a closed room, taking advantage of the light's confineability feature. The bootstrap phase is followed by a more classical key-exchange phase, where the actual key material gets exchanged in the presence of active attacks – relying on the shared states from the bootstrap phase. We formally define this new type of key-exchange protocol which offers authenticated key exchange with post-compromise security without relying on pre-shared secrets. We then show that a simpler and more efficient version of the signed Diffie-Hellmann protocol, now relying on MACs instead of signatures for the mutual authentication, can be proven secure in our model. Finally, a proof-of-concept implementation of the SEKA protocol is evaluated in a testbed demonstrating the efficiency gains of our approach.

1. Introduction

Light Fidelity (LiFi) networks constitute an alternative or addition to existing Radio Frequency (RF) networks. They rely on light waves as carrier waves, utilizing Light Emitting Diodes (LEDs) as transmitters and photocells as receivers. Different modulation schemes are used to encode digital information onto light waves, providing different trade-offs between energy efficiency and data rate. For example, the IEEE 802.15.13 LiFi standard defines two physical layers that utilize On-Off-Keying favoring energy efficiency and Orthogonal Frequency Division Multiplexing (OFDM) favoring data rate, respectively [19]. An exemplary LiFi network, serving a factory hall, is illustrated in Figure 1.

In comparison to Radio Frequency networks that rely on radio waves, LiFi networks provide performance and reliability advantages. As LiFi network are physically confined by walls, inter-cell interference solely needs to be considered within the same room. Therefore, in terms of performance, LiFi networks can provide data rates between 100 and 1000 times faster than RF networks [34].

Security via Confineability. Being able to block the propagation of LiFi signals in certain directions, is also a core advantage over RF networks in terms of security. In particular, as light waves are fully absorbed or reflected by opaque materials, LiFi networks are unable to escape a fully closed room (*confineability*). This is not generally true for RF networks, whose carrier waves are easily capable of traveling large distances and through obstacles.

This combination of high throughput and natural security properties, makes LiFi highly suitable for industrial and medical communication systems. Therein, LiFi is used to establish local networks among a group of neighboring devices. While cryptography is used to protect communication in these networks, the underlying key exchange often relies on the physical protection alone [29–31, 33, 35, 36, 39, 44, 45]. For example, Suduwella et al. [36] propose transfering the WPA2-PSK secret key in plain via LiFi within a confined space in order to allow devices to connect to the network conveniently.

Despite its strong characteristics on the physical layer, security should not solely rely on the confineability of signals. An empirical study by Classen et al. [10] found that eavesdropping on LiFi networks using indirect light paths, e.g., through keyholes or door gaps, allows attackers to decode a large part of transmitted messages, even at high data rates. As a consequence, several of the aforementioned schemes ([29–31, 33, 39, 44, 45]) utilize jamming and other techniques in order to artificially decrease the signal-to-noise-ratio of any eavesdropper, decreasing the likelihood that the attack by Classen et al. succeeds. However, the security of this approach relies on strong,

§. Research conducted while affiliated with HPI and Fraunhofer HHI. Eric Ackermann is a member of the Saarbrücken Graduate School of Computer Science.

DOI 10.1109/EuroSP60621.2024.00041

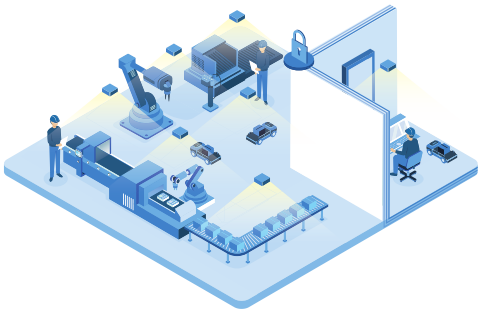

Figure 1: Industrial LiFi network with ceiling-mounted optical front ends (OFEs) and autonomous vehicles as endpoints. A maintenance room (on the right) is isolated from the light propagation in the main hall (on the left).

novel assumptions about LiFi channels that have not been studied extensively [2, 9, 47]. In particular, physical-layer-security protocols like [30, 36] transmit secrets in the clear and thus are insecure in our model. Their security is based on the assumption that the eavesdroppers' location is precisely known, such that their received signal can be attenuated enough that the adversary is unable to decode the communication.

Basing security on uncertain physical properties will only lead to an arms race in attack and defense mechanisms. Luckily, establishing shared key material over an insecure channel is one of the most fundamental – and well solved – problems in cryptography. That is, any secure (authenticated) key exchange protocol can be used in a LiFi network in order to exchange keys and bootstrap secure communication.

Password-based Key Exchange in LiFi. In fact, recent LiFi standards such as IEEE 802.11bb [18] or Rec. ITU-T G.9991 [22] rely on channel-agnostic key exchange protocols instead of physical security. Both use password-authenticated key exchange protocols (PAKEs), namely Dragonfly [41], in the case of the IEEE 802.11bb LiFi network, and PAK [7], in Rec. ITU-T G.9991.

Thus, the authentication in the key exchange protocol requires choosing and manually entering strong pass phrases into the devices. This is a tedious and error-prone process, especially when the password is complex. Also, by design, security of a PAKE hinges on a proper selection and usage of passwords. It is easy to violate this assumption, e.g., by writing the password down, choosing weak passwords or relying on weak default passwords [26, 43, 48].

Key-based Authentication. Key exchange protocols such as EAP-TLS rely on public-keys for mutual authentication instead of passwords and clearly provide superior security to any password-based approach. The additional certificate verification increases the computational costs of the key exchange protocol though. Furthermore, bootstrapping the initial keys and certificates bears several challenges and none of the existing solutions truly fits the industrial communication setting where autonomous devices aim to establish a local network.

A typical way to establish trust, is to equip devices with manufacturer-installed and device-specific keys, certified by a trusted entity. For the targeted domain of industrial communication systems, this makes the network susceptible to *wardriving* attacks: An adversary who has ownership of any industrial device with a valid manufacturer-supplied key can attempt to associate the device with *any* industrial network, using the access for attacks against the legitimate industrial devices and other hosts in the network. As especially industrial internet of things (IIOT) devices are becoming increasingly affordable, the entry barrier for this attack is becoming lower and lower. Thus, a solution for verifying whether a device may join the network is still required. This increases operating costs and requires a vigilant human operator.

Messaging protocols such as Signal utilize a *semi-trusted server* instead of a PKI to address this issue [12]. User clients upload long-term (and ephemeral) keys to the server, and rely on the server for secure provisioning of the adequate keys. Apart from the strong trust that needs to be put in the central server, another pitfall of this solution is that clients require an *authenticated channel* with the server, as they cannot verify the authenticity of the key material. In contrast to instant messaging that runs over a well-established internet PKI, this solution is not a good fit for network authentication. In order to ensure network availability, authentication servers are typically set up on-premise. In such a setting, utilizing the Signal protocol would only shift the problem from establishing a secure channel between a network member and infrastructure devices to establishing a secure channel between network member devices and the semi-trusted server, again requiring manual configuration or PKI.

In summary, none of the existing key exchange protocols provide the ease-of-use that would be needed to establish secure self-contained LiFi networks comprised of numerous local physical devices.

1.1. Our Contributions

In this work we investigate whether the channel characteristics of LiFi networks can be leveraged for the construction of a key exchange protocol that provides a better trade-off between security, performance and ease-of-use for the industrial LiFi domain than existing approaches. In particular, both the human interaction required and the necessary setup assumptions should be minimal, and at the same time no unreasonable assumptions on the physical protection should be made.

This paper solves that challenge by introducing the Secretless Key Exchange and Authentication (SEKA) protocol, an authenticated key exchange protocol for LiFi networks. SEKA provides provably secure and highly efficient key exchange in the presence of (partially) *active* adversaries in the LiFi setting *without* relying on pre-installed secrets such as private keys or passwords.

The core idea is to split the key exchange into two different phases: a bootstrap and a key exchange phase. In the bootstrap phase we assume that special care is taken to take advantage of the *confineability* of LiFi networks. That is, the devices to exchange keys via LiFi are put in a closed and protected room, such as the maintenance room in Figure 1. We do not want to fully rely on the physical protection, and generously assume that an adversary can eavesdrop on all communication. We do exclude active

capabilities though, and restrict the adversary to passive behavior in this bootstrap phase. The devices will use the closed and protected area to establish a pairwise secret state which each other.

This shared secret is then utilized to exchange – and regularly update – shared keys in a more standard key exchange phase. Thereby, the LiFi devices can be moved out of the separate room, as this phase can provide security against *active* adversaries. A core technical step needed here is to realize that "authentication-by-recognition", i.e., devices being able to securely recognize that they are communicating to a particular device they trusted at an earlier stage, is sufficient for the targeted network setting.

In more detail, we make the following contributions:

Formal Model. Our core contribution is to reflect and leverage the stronger physical features that LiFi can provide in a formally sound way, and confining the stronger assumptions to a short phase of the protocol. We provide a formal definition and security model for SEKA, capturing *passive* eavesdropping during the *Bootstrap* phase and *active* attacks (including, e.g., message modification, impersonation and unknown key share attacks) in all subsequent protocol executions. Our model is based on the state-of-the-art security model for key exchange by Cohn-Gordon et al. [11], and even allows compromise of random number generation and all device secrets, under the assumption that the adversary stays passive in *at least one session* after the compromise. This is the first model, that captures Post-Compromise Security (PCS) for protocols that do not rely on pre-installed secret keys.

Simple and Efficient Protocol. We then propose a simple and provably secure construction of SEKA from generic building blocks, and a *minimal* number of NIST-standardized concrete instantiations. The protocol follows the classical Diffie-Hellman based key exchange [13], but uses symmetric authentication instead of signatures and public-keys for the mutual authentication – improving the efficiency compared to the standard approach.

Implementation and Benchmarks. We provide a reference implementation of SEKA and a computation-time benchmark of SEKA in comparison to the WPA key exchange protocols, showing that SEKA outperforms the WPA security protocols by factors between 10 and 100. Finally, an evaluation of the suitability of SEKA for Time-Sensitive Networking (TSN) and multi-gigabit throughput, shows that SEKA provides the performance required for real-world LiFi networks.

1.2. Related Work

Key exchange protocols that do not rely on out-of-band channels or pre-installed secrets have already been deployed in wireless networks, e.g., Wi-Fi Protected Setup (WPS) [1], the "just works" mode of Bluetooth SSP [6] and Opportunistic Wireless Encryption (OWE) with the optional PMK Caching feature [17]. In case the host keys are not compared out-of-band, SSH effectively also *functions* as SEKA protocol [46]. Like SEKA, these protocols consist of an initial phase that negotiates an authentication key followed by an authenticated key exchange. However, none of the existing protocols provide SEKA security:

OWE uses a textbook (non-authenticated) Diffie-Hellman protocol to establish a mutual shared secret authentication key. This key is then used as pairwise master key in a second phase, the 4-Way Handshake, which resembles the *rekey* protocol proposed by Canetti and Krawczyk [8]. In particular, symmetric primitives are used for mutual authentication and key derivation. However, OWE only provides security against passive adversaries and does not guarantee post-compromise or forward security as demanded by our SEKA security model: An active adversary with MITM capabilities can learn the initial authentication key. The compromise of the internal state invalidates the security of all previous sessions as the adversary can re-compute *all* session keys from the transcript. Likewise, as long as the authentication key remains valid, the security cannot be regained, even if the adversary remains passive in all future sessions. Finally, the lifetime of the bootstrapped authentication key is unspecified. After its expiry, the initial bootstrap phase needs to be executed again *without authentication*, enabling attacks against authentication like evil twin.

Bluetooth SSP in the "just works" mode is conceptually identical to OWE. The main difference is that in SSP, the initial authentication key has an unlimited lifetime. Thereby, given a passive adversary during the initial phase, SSP can provide mutual authentication. Like OWE, SSP cannot recover from MITM attacks during the initial phase. Adding insult to injury, downgrade attacks allow an adversary to force two devices to fall back to the "just works" mode when stronger authentication, e.g., using short authenticated strings, would be available. Thus, the conceptually more secure SSP modes can be attacked using the generic MITM attack against the Diffie-Hellman key exchange [16, 37, 38].

WPS computes an authentication key in the same way as OWE and uses it for encrypted transfer of the network password used by the WPA protocols. However, as the network password is the only form of authentication in WPA (personal), successful attacks against WPS can lead to the adversary learning the passphrase [42]. The WPA/WPA2 protocols are not designed to provide *any* security for *any device in the network* after the password has been compromised [23, 40]. In fact, the 4-Way Handshake underlying the WPA2 protocol does not provide forward secrecy, which Canetti and Krawczyk pointed out for their conceptually identical "Rekey" protocol [8]. The more recent WPA3 protocol provides PFS, i.e., it can provide security against *passive* adversaries and for previous communication after key compromise [24, 41].

SSH can initially exchange public host keys in-band or out-of-band, enabling an authenticated DH in the second phase. Thereby, SSH can provide forward secrecy [8, 24]. However, as it is stateless, it cannot provide post-compromise security [11]. Thereby, again, it cannot recover from MITM attacks during the initial exchange of authentication keys, even if the adversary remains passive in a later session.

Section 5.2 discusses the benefits that SEKA can provide in comparison to the existing protocols.

2. SEKA – Functionality & Security

This section introduces the new type of secret-less key exchange (SEKA), and defines the desired security properties through a formal security model. We start with the functionality and syntax, and then explain and thoroughly define the guaranteed security properties.

Notation. Throughout this paper, we use the following notational conventions. Assignment of a variable will be denoted as follows: $a \leftarrow b$. Uniformly random assignment of a variable will be denoted like $a \leftarrow_R \{0,1\}$. Binary concatenation of two values a and b will be denoted like $a||b$. Undefined or invalid values will be denoted as $\perp$, ignored output values will be denoted as _ and ignored or arbitrary input values will be denoted as the $\cdot$ symbol. τ is the security parameter.

2.1. Definition of SEKA

SEKA is a novel two-party key exchange protocol that was designed specifically for LiFi networks, and is intended for mobile or embedded LiFi devices without input/output capabilities. SEKA is executed in two phases, Bootstrap and Key-Exchange.

Bootstrap is expected to be executed in a physically secure, confined location, and generates an initial pairwise authentication secret (called the *protocol state*).

The Key-Exchange phase uses the current protocol state for mutual authentication of the parties and for the exchange of a session key between the parties in the presence of an active adversary. In each invocation, the party that transmits the first message assumes the *initiator* ($\mathcal{I}$) role, and the party that receives the first message assumes the *responder* ($\mathcal{R}$) role.

More formally, SEKA is defined as follows:

Definition 1 (SEKA Protocol). A SEKA protocol is a tuple (Initialize, Bootstrap, Key-Exchange, $\mathcal{K}$) of a probabilistic function Initialize, two-party protocols Bootstrap and Key-Exchange and a probability distribution of session keys $\mathcal{K}$ such that:

Initialize$(1^\tau) \rightarrow pp$ On input the security parameter τ, returns the public parameters pp.

Bootstrap : $\langle P_i(pp, ID_i, ID_j, role = \mathcal{I}); P_j(pp, ID_j, ID_i, role = \mathcal{R})\rangle \rightarrow st_{i,j}, st_{j,i}$ A two-party protocol between a party P_i with identity ID_i[1] in initiator role and a party P_j with identity ID_j in responder role. After it was accepted[2], the initial state $st_{i,j} = st_{j,i}$ is output for both parties.

Key-Exchange : $\langle P_i(pp, st_{i,j}, ID_i, ID_j, role = \mathcal{I}); P_j(pp, st_{j,i}, ID_j, ID_i, role = \mathcal{R})\rangle \rightarrow (k_i, st'_{i,j}), (k_j, st'_{j,i})$ A two-party protocol between parties P_i with identity ID_i in initiator role and P_j with identity ID_j in responder role. Its input $st_{i,j} = st_{j,i}$ is the shared secret state output by the last invocation of Key-Exchange or Bootstrap initially, the other inputs have the same semantics as in Bootstrap. The output is a session key $k_i = k_j$ with $k_i \in \mathcal{K}$ and a state $st'_{i,j} = st'_{j,i}$.

1. In the implementation, ID_i and ID_j are 48-bit MAC addresses.
2. Accepted refers to *completed and not aborted* in this context.

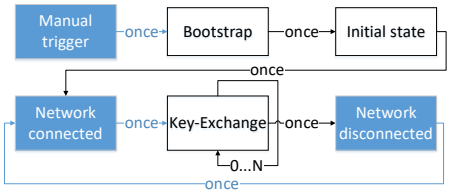

Figure 2: Intended order of the SEKA phases.

The execution order of Initialize, Bootstrap and Key-Exchange is as follows. Initialize is run first and only once. Bootstrap is run second, and also only once. Finally, Key-Exchange can be run arbitrarily often. The output state of each invocation of the protocols is passed to the following invocation of any of the protocols. If any output is $\perp$, the protocol is considered *aborted*.

The order of the SEKA phases is illustrated in Figure 2.

We require any SEKA protocol to be correct. Informally, Bootstrap is correct if two honest parties compute the same output state. Key-Exchange is correct if two honest parties compute the same session key and output state. More formally, correctness is defined as follows:

Definition 2 (SEKA Correctness).
A SEKA protocol is correct when Bootstrap and Key-Exchange fulfill these requirements:

- The Bootstrap protocol is correct if all honest parties P_i, P_j that invoke the protocol with roles $\mathcal{I}$ and $\mathcal{R}$ respectively and the same identities ID_i, ID_j and public parameters pp output the same state $st_{i,j} = st_{j,i}$ and accept.
- The Key-Exchange protocol is correct if all honest parties P_i, P_j that invoke the protocol with the same state $st_{i,j} = st_{j,i}$ and roles $\mathcal{I}$ and $\mathcal{R}$ respectively and the same identities ID_i, ID_j and public parameters pp output the same state $st'_{i,j} = st'_{j,i}$ and session key $k_i = k_j$ and accept.

2.2. Security of SEKA

This section starts by stating the security goals of SEKA informally. After that, key terms and concepts are introduced before the formal security definition is stated.

Because SEKA is designed for link layer security protocols rather than application-layer security, its design goals partially differ from existing security models [8, 11, 25], and are as follows:

Bootstrapped Authentication. First, in the interest of ease-of-use, SEKA is *secretless*. This means initiator and responder initially have no information about each other. In particular, neither a pre-shared secret nor a public-key infrastructure is assumed. Instead, when a pair of parties communicate for the first time, they need to *bootstrap* a shared secret, the *initial state* st. This is why the first phase of the protocol is called Bootstrap. The pairwise state st is the only long-term secret in the SEKA protocol. The state is then used to authenticate messages during the second phase, Key-Exchange.

Authentication by Recognition. Second, authentication is only provided implicitly. If the Key-Exchange phase commences between two parties identified by MAC addresses after the same parties accepted Bootstrap, they *recognize* each other using the shared state. In other words, authentication in the context of SEKA means that devices are ensured that they are still communicating with the same device they performed Bootstrap with. This level of authentication is sufficient for protecting LiFi networks from common attacks against link-layer protocols such as eavesdropping, MAC address spoofing and wardriving.

Secure Key Exchange and Post-Compromise Security. Clearly, the main goal is to provide secure key exchange over an insecure channel. That is, an adversary being able to intercept (or even modify) all exchanged messages between two honest parties must not learn anything about the exchanged key. The adversary is limited to passive attacks in the Bootstrap phase, but can actively interfere with all exchanged messages in the Key-Exchange sessions. Our protocol guarantees *Forward Security* in the Key-Exchange phase, i.e., if a party gets eventually compromised in a session, and the adversary learns its full state, the security of the previous sessions is not affected.

Further, SEKA also provides *Post-Compromise Security* for the Key-Exchange phase. This means that the protocol is able to *heal* from the compromise, leaving the adversary with no advantage in guessing the session key in future sessions. To achieve this strong property, the protocol again requires passive behavior of the adversary in at least one session after the last compromise. This is the same as in the work by Cohn-Gordon et al. [11], and we call such a session with passive adversarial behaviour only, a *clean* session. In the context of LiFi, ensuring a clean session can be realized e.g., by moving the devices in a closed room again or removing all untrusted devices from the communication area. After the clean session was accepted, the protocol provides the same security for the following sessions as if there had been no compromise, i.e., the compromise is healed. In fact, a clean session in the Key-Exchange phase can also heal an active attack in Bootstrap. As the Bootstrap session is intended to be a clean session by design, we do not focus on that option in the rest of the discussion though.

In summary, our protocol guarantees secrecy of exchanged keys in the presence of active attacks, if there is at least one clean session before (typically the Bootstrap) and can even heal from compromise if another clean session is completed after the compromise. That is, SEKA requires one clean session between each pair of parties *regardless* if they were ever compromised. This uses (a weaker variant of) the *confineability* assumption made about the underlying LiFi channel. In other words, while Cohn-Gordon et al. use passivity of the adversary only to *recover* from out-of-band attacks such as device compromise, a passive adversary during at least one session is a central design element of the SEKA security model and the reason SEKA does not require out-of-band-channels or pre-installed secrets.

In Section 5.1 we discuss why be believe that the limitation of passive adversarial behavior is a reasonable assumption based on the confineability of light.

2.2.1. Security Model. We now present our formal security model that captures the aforementioned properties, Our model closely follows the existing key exchange models, such as [4, 8, 11, 24, 25]. Roughly, it gives the adversary oracle access to honest parties with which it can interact through the stages, and even allows to compromise them. Eventually, there is a test session in which the adversary is tasked with the challenge of distinguishing a random key from the actual session key established by two honest parties. The main challenge in the model is to keep track of the adversaries behavior and detect and exclude any trivial wins. In more detail, our model makes use of the following concepts and modelling choices:

Sessions. An invocation of SEKA between two parties leads to an exchange of messages or *conversation*. Each party has its own view on a conversation or *session* consisting of the messages it transmitted and received. If two sessions are two parties' views on the same conversation, they are *matching*. Since the adversary controls message delivery, a session *need not* have a matching session. Sessions have an *ephemeral state* as defined in Table 1.

Oracles. The adversary has access to the oracles listed in Table 2, which allows it to interact with honest parties, trigger their behaviour or compromise them. The oracles used in the SEKA security model are based on the oracles in the Post-Compromise Security model [11]. The oracles allow the compromise of both long-term and ephemeral secrets, session keys, generated random numbers and insertion, deletion and modification of messages. Unrestricted utilization of the oracles enables trivial victory against any valid SEKA construction, as the adversary is able to, e.g., *query* the session key that it is expected to *guess*. Thus, in the next step, a number of *freshness predicates* are defined. These limit the permissible use of the oracles in a way such that no trivial victory is possible. Also, the restriction that the adversary needs to behave *passively* during a session (which is *intended to be* Bootstrap, but *can also be* Key-Exchange) is implemented in the freshness predicates.

Freshness Predicates. Finally, a set of *freshness predicates* is defined, constraining the adversary's access to the oracles such that trivial victory is not possible. Adherence to the freshness predicates is enforced by requiring the *test* session to be *test-fresh* in the corresponding oracle.

The freshness definitions use the definition of *matching and partially matching sessions* provided by Cohn-Gordon et al.: Two sessions are *partially* matching if both parties agree on the identity and role of their respective peer and all except possibly the last message in the conversation. Two sessions are *matching* if they are partially matching and the parties agree on *all* messages exchanged in the session [11]. After giving formal definitions for the freshness predicates, it is detailed how the security properties are reflected in the definition.

Definition 3 (Fresh sessions). A session s identified by (ID_i, ID_j) and $invocation\text{-}id_s = i$ in P_i is considered *bootstrap-fresh* if none of the following oracle queries were made by the adversary:

- $randomness(ID_i, i)$ or $cr\text{-}create(ID_i, ID_j, \cdot)$ creating s

Attribute	Description
$party_s$	Identity of the party at which s was created by the creation oracles or by receipt of the first message.
$role_s$	Role of the party in s, either $\mathcal{I}$ (*initiator*) or $\mathcal{R}$ (*responder*). The initiator is the party which is invoked from the outside (i.e., the adversary in the security game or the operator), while the responder is the party which receives the first message from the initiator.
$randomness_s$	All random inputs that were read from the random tape of the party during the execution of s.
$initiator\text{-}identity_s$, $responder\text{-}identity_s$	Identity of the initiator and the responder of s, respectively.
π_s	The protocol phase that is executed by s, $\pi_s \in \{\mathsf{Bootstrap}, \mathsf{Key\text{-}Exchange}\}$.
$status_s$	Current status of s, $status_s \in \{unknown, active, aborted, accepted\}$. Sessions are initially $unknown$. When they are invoked at a party, they become $active$. When the last protocol message for the session s has been sent or received at the party, respectively, and an output different from $\perp$ is generated, the session transitions into $accepted$. When an $active$ session is explicitly aborted by the protocol, the state transitions to $aborted$. Sessions are $completed$ when they are either $accepted$ or $aborted$.
$sent_s$, $received_s$	Concatenation of all messages sent or received by s, respectively.
$invocation\text{-}id_s$	The session was the $invocation\text{-}id_s{}^{\text{th}}$ session that was created at $party_s$.

TABLE 1: State attributes of a SEKA session s known to a party P_i.

Oracle	Description
$cr\text{-}create(ID_i, ID_j, role, \pi, rnd) \rightarrow m_1$	"Corruptly" create new session with initiator identity ID_i and responder identity ID_j, subprotocol $\pi \in \{\mathsf{Bootstrap}, \mathsf{Key\text{-}Exchange}\}$, role $role$ and attacker-supplied random tape rnd, passing the input to the protocol accordingly and return first message m_1. rnd is used for all computations until the session is completed. To this end, the oracle starts the honest party P_i with access to the supplied random tape rnd and its persistent database of states $st_{i,j}$ and activates either Bootstrap or Key-Exchange, returning the first protocol message if $role = \mathcal{I}$. For all parties and possible peers, a session that corresponded to the Bootstrap phase must be accepted *before* the first Key-Exchange phase can be created. Also, at most one Bootstrap phase may be created for each pair of parties.
$create(ID_i, ID_j, role, \pi) \rightarrow m_1$	Invoke *cr-create* with rnd sampled from an ideal random number source, i.e., create an honest session.
$send(ID_i, i, m_{in}) \rightarrow m_{out}$	Process message m_{in} in the i^{th} session started by honest P_i, return m_{out} or $\perp$ if there is no response.
$corrupt(ID_i) \rightarrow (st_{i,j}, st_{i,j'}, ...)$	Return all states $(st_{i,j}, st_{i,j'}, ...)$ in the persistent state database of party P_i. The tuple is empty initially.
$randomness(ID_i, i) \rightarrow randomness_i$	Reveal $randomness_i$ of the i^{th} session in party P_i, i.e., all outputs of the random number generator during the session (see Table 1). To this end, $\mathcal{A}$ gains read access to the random tape used in the session.
$session\text{-}key(ID_i, i) \rightarrow k$	Reveal session key k of the i^{th} Key-Exchange session by honest P_i. The session must be *accepted*.
$test\text{-}session(ID_i, i) \rightarrow k'$	Generate a random bit $b \leftarrow_{\text{R}} \{0,1\}$ and if $b = 1$ return $k' \leftarrow session\text{-}key(ID_i, i)$, else return random $k' \leftarrow_{\text{R}} \mathcal{K}$. The session s with $invocation\text{-}id_s = i$ must be suitable for $session\text{-}key$ and be *test-fresh*.
$guess(b')$	Terminate with guess b'.

TABLE 2: Oracles available to the adversary in our security model, adapted from [11].

- if a session s' partially matching s in P_j with $invocation\text{-}id_{s'} = i'$ exists, either $randomness(ID_j, i')$ or $cr\text{-}create(ID_j, ID_i, \cdot)$ creating s'

A session s identified by (ID_i, ID_j) and $invocation\text{-}id_s = i$ in P_i is considered *test-fresh* if none of the following oracle queries were made by the adversary:

- $corrupt(ID_i)$ between the completion of the *clean* session and any partially matching sessions and the completion of the session s and any partially matching sessions
- $corrupt(ID_j)$ between the completion of the *clean* session and any partially matching sessions and the completion of the session s and any partially matching sessions
- $corrupt(ID_j)$ after the completion of the *clean* session and any partially matching sessions if no partially matching session to the test-session exists or it was not accepted.
- $session\text{-}key(ID_i, i)$ revealing the session key of s
- $session\text{-}key(ID_j, i')$ revealing the session key of s' if s' partially matches s.

and a *clean session* exists.

A session c is considered *clean* if it is the *last* session such that:

1) There is a partially matching[3] session c'.
2) c and c' were *accepted* before the test session or any of its partially matching sessions were *created*, or c *is* the test session and both c and c' were *accepted*.
3) The partners of the clean session are also the partners of the test session.
4) c is *bootstrap-fresh*.
5) If c executes Bootstrap, c' is also *matching*, otherwise, c' is at least *partially matching*.

The notion of *bootstrap-freshness* in the definition captures a session whose *randomness* has not been corrupted by the adversary. A *clean* session is a session during which the adversary remained passive, i.e., relayed messages faithfully ((*partially*) *matching*) and did not corrupt randomness (*bootstrap-fresh*). A session is *test-fresh* when it is preceded by a *clean* session *between the same pair of parties* and the adversary did not query the pairwise *state* after the completion of the *clean* session and did not query the session key of the *test-fresh* session. The adversary needs to distinguish the session key of the *test-fresh* session from random in the security game.

Robustness. SEKA also relies on the *robustness* definition by Cohn-Gordon et al. [11]. Informally, if a protocol

3. Remember that this means both matching peers and roles and unmodified messages.

is *post-network robust*, no adversary that is restricted to message modification is able to *desynchronize two parties*, i.e., cause the state st stored in both parties not to match. If this was not given, adversaries could prevent parties from communicating using only message-modification attacks.

Security Game. SEKA uses the security game defined by Canetti and Krawczyk [8]. With $\mathcal{O}$ being the tuple of oracles defined above, $st_{\mathcal{A}}$ being an optional adversarially-defined state, $fresh(ID_s, s) \longrightarrow f$ being a deterministic function that returns 1 if session s in party ID_s is *test-fresh* according to the freshness predicates defined above and 0 otherwise, $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ being an adversary comprised of two PPT adversaries and $\mathcal{K}$ being the protocol-specified session key probability distribution, the security game can be stated:

$pp \leftarrow \mathsf{Initialize}(1^\tau)$
$ID_i, s, st_{\mathcal{A}} \leftarrow \mathcal{A}_1^{\mathcal{O}}(pp)$
$k_0 \leftarrow_{\mathrm{R}} \mathcal{K}, k_1 \leftarrow session\text{-}key(ID_i, s), b \leftarrow_{\mathrm{R}} \{0,1\}$
$b' \leftarrow \mathcal{A}_2^{\mathcal{O}}(pp, st_{\mathcal{A}}, k_b)$
if $b = b' \wedge fresh(ID_s, s) = 1$ **output** 1
else output 0

Note that the session s returned by $\mathcal{A}_1$ is the session provided to the *test-session* oracle, and the bit b' returned by $\mathcal{A}_2$ is the bit provided to the *guess* oracle. Also, remember that the session s must be *test-fresh*.

Definition 4 (Secure SEKA protocol). Let $n_p \in \mathbb{N}, n_s \in \mathbb{N}$ be the maximum number of parties and sessions per party, respectively. A SEKA protocol π is said to be secure when for all n_p, n_s all of the following requirements hold for all PPT adversaries $\mathcal{A}$:

1) The advantage of $\mathcal{A}$ in guessing the bit b, i.e., the probability that $b = b'$, is equal to or less than $\frac{1}{2} + negl(\tau)$.
2) π is *post-network robust* if network adversaries are restricted to message-passing during invocations of Bootstrap.

2.2.2. Discussion of our Model. We now explain how the security properties sketched at the beginning of the section are realized in this model.

Bootstrapped Authentication via Freshness Predicates. Informally, the *clean* session serves bootstrapping authentication between two parties. Therefore, the adversary is restricted to passive eavesdropping (by demanding a *matching* session) and prohibited from revealing randomness used in the *clean* session, and a *clean* session must precede the *test* session. Thereby, the adversary has no trivial knowledge of the output state and session key of the *clean* session, giving the protocol the opportunity to compute a secret shared state for use in later sessions. This is why SEKA does not require pre-installed keys. Note that the *clean* session may execute either protocol phase, modelling both a successful initial Bootstrap phase and *recovery* from the compromise in Key-Exchange.

Authentication by Recognition via Freshness Predicates. Also, the adversary is not allowed to query the session key of the *test* session or any *partially matching* session. Obviously, allowing the adversary to query the session key of the *test* session leads to trivial victory. Due to protocol correctness, the partially matching session is *expected* to output the same session key if it accepts, which is why its session key cannot be queried either.

Note that if the adversary manages to create a session that computes the same session key as the *test* session *but does not partially match it*, the adversary is allowed to query the session key of that session and to win the game. This is a valid attack against the *Authentication by Recognition*, and such a session should not be accepted, such that the $session\text{-}key$ oracle cannot be used on it.

Modelling of Post-Compromise Security. As in the original Post-Compromise Security model presented by Cohn-Gordon et al., in the proposed model, the SEKA adversary is allowed to compromise all state of the peers of the test-session *before* the test-session is created [11]. Also, as in the previous work, a clean session is required to be accepted before the test-session, and no state compromise between the start of the clean session and the expiration of both the clean session and the test-session is allowed. Additionally, the adversary is restricted to passivity in the clean session. Contrary to the original Post-Compromise Security model, in the SEKA model, the clean session and the test session may be the same session. In this case, the adversary is restricted to passing messages honestly and prohibited from querying the ephemeral state of the session, however. Thereby, this case on its own is equivalent to the weak perfect forward secrecy definition by Krawczyk, covering the scenario of a fully passive adversary during the test session [24].

The main differences between SEKA and the original Post-Compromise Security model are related to authentication and the assumptions about long-term states. Cohn-Gordon et al. use explicit (strong) authentication, relying on public keys for both initiator and responder to be exchanged over a tamper-resistant out-of-band channel [11]. On the other hand, the SEKA provides Bootstrapped Authentication and Authentication by Recognition without relying on ouf-of-band channels. Therefore, the $corrupt$ oracle was changed to only return the long-term state st, as no long-term key exists. Finally, for the SEKA security model, the restriction of passive adversarial behavior during the Bootstrap phase in the security and robustness game is added. There is no equivalent requirement in the Post-Compromise Security model since it relies on the exchange of public keys via an out-of-band channel [11].

3. Our SEKA Protocol

We now present our simple construction of a SEKA protocol. In a nutshell, our protocol follows the classic approach of Canetti and Krawczyk [8]. It combines a passively-secure two-round key exchange protocol (KE) with a Message Authentication Code (MAC). The Bootstrap phase excludes active attacks, and runs KE to establish a key from which both parties derive a shared MAC key. In the Key-Exchange the actual key material k gets derived in the presence of active adversaries and even state compromise. Here we again use the passively-secure KE but now authenticate the exchanged messages via the MAC keys from the Bootstrap phase. For each round, a

new key k and state (serving as MAC key in the next round) is derived from the current state and the output of KE via a key-derivation function. The detailed description is in Figure 3 and Figure 4 and discussed below.

The construction of the Key-Exchange phase takes inspiration from previously proposed constructions. The signature-authenticated Diffie-Hellman key exchange proposed by Canetti and Krawczyk is used as the template for how we use a MAC scheme to authenticate the messages of the underlying passively secure key exchange protocol in Key-Exchange [8]. Furthermore, Cohn-Gordon et al. have proposed a transformation that can be used to add post-network robustness and post-compromise security to *any* authenticated key exchange with security under the AKE security model that was previously proposed by LaMacchia et al. [11, 25]. This transformation has inspired the encoding and utilization of the pairwise state in SEKA.

Building Blocks. The construction of the SEKA requires the following building blocks. For space reasons we introduce them only briefly, and refer to Appendix A for the detailed definitions.

Key Derivation Function (KDF), i.e., a function $\mathsf{KDF}(\sigma, l, r, c) \longrightarrow k$ that deterministically computes a key k of l bits from keying material σ with salt r and (optional, arbitrary) context c. In the security proof, the KDF is assumed to be a random oracle.

Message Authentication Code, i.e., a MAC $(\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$, where we assume $\mathsf{KGen}(1^\tau)$ to randomly sample keys $k \leftarrow \{0,1\}^\tau$, and tags are generated via $t \leftarrow \mathsf{Mac}(k, m)$ and verified via $0/1 \leftarrow \mathsf{Vf}(k, m, t)$ respectively. We require the MAC to be deterministic and have $SNF^{CMA}_{\mathcal{A},\Pi}$-security. Determinism is necessary, since the adversary in our model is allowed to corrupt and influence random number generation during the *test* session and common MAC security definitions do not consider compromise of the random number generator. Note that Game 5 in our security proof relies on strong unforgeability of the MAC scheme.

Two-round key exchange protocol with CK^{AM}_{PFS}-security [8] and weak perfect forward secrecy [24], i.e., the key exchange is expected to provide key indistinguishability against a *passive* adversary *before and after* compromise of long-term secrets. The key exchange protocol has an initialization function $\mathsf{KE.init}(1^\tau) \longrightarrow pp$ and two further interfaces: $\mathsf{KE.exchange}(pp, ID_i, ID_j, s, role) \longrightarrow k$ is the uninterrupted run of the key exchange protocol for parameters pp, own identity ID_i, peer ID_j, session ID s and role $role \in \{\mathcal{I}, \mathcal{R}\}$. It performs the key exchange with the peer and returns a session key $k \in \{0,1\}^\tau$. We will use this interface in the Bootstrap-phase of our protocol. $\mathsf{KE.handle}(pp, ID_i, ID_j, s, role, st_{eph}, m_{in}) \longrightarrow (st'_{eph}, m_{out}, k)$ is the corresponding message-by-message interface to the key exchange protocol and will be used in the Key-Exchange sessions.

State Conventions. Both phases encode the pairwise state $st_{i,j}$ as a tuple (st_{i,j_c}, st_{i,j_p}) of the current state st_{i,j_c} and the list of potential states st_{i,j_p}. This separation has been proposed by Cohn-Gordon et al. and is necessary for post-network robustness. Section 3.1 explains how st_{i,j_p} is used to resolve situations in which the current states of two parties st_{i,j_c}, st_{j,i_c} differ. This can happen when the third message of Key-Exchange is not delivered to the responder. Without resolution, in this scenario, the parties would be unable to ever communicate again [11].

The security model makes no assumption regarding the structure of the pairwise state $st_{i,j}$. Therefore, the *corrupt* oracle reveals *both* parts of the pairwise state, i.e., both current and potential states, to the adversary. Thus, the protocol needs to carefully remove states that are no longer needed from st_{i,j_p} in order to achieve security under the security model.

3.1. Protocol Description of π_{SEKA}

Our π_{SEKA} protocol is initialized by running the corresponding algorithm from the underlying key exchange, i.e., $\mathsf{Initialize}(1^\tau)$ returns $pp \leftarrow \mathsf{KE.init}(1^\tau)$. The detailed Bootstrap phase is given in Figure 3 and Key-Exchange is specified in Figure 4.

Bootstrap. The construction simply performs an initial key exchange over the insecure channel. Remember that the adversary is assumed to remain passive during Bootstrap. Thus, CK^{AM}_{PFS}-security of the underlying key exchange (i.e., security against passive adversaries) is sufficient here and guarantees that the adversary cannot distinguish the output state from random. The KDF is then used to derive an τ-bit initial state from the exchanged key. The salt in the KDF is set to the nonce s. This ensures domain separation and is necessary for the provable indistinguishability from random of the initial state, as discussed by [15]. Also, the context is used to associate the initial state with the identities of the parties, which is necessary to lay the foundations for the Authentication by Recognition and Bootstrapped Authentication.

Key-Exchange. This phase runs a stateful authenticated key exchange composed of a passively-secure key exchange KE and a MAC. In essence, the signed Diffie-Hellman protocol (DH is here generalized as a generic key-exchange protocol KE) presented by Canetti and Krawczyk [8] is adapted to the symmetric setting, replacing the signature scheme with a MAC. The MAC uses the current pairwise state st_{i,j_c} as key, as in the transformation presented by Cohn-Gordon et al. [8, 11]. The MAC is used to authenticate all exchanged KE values an honest party ID_i sends to ID_j (and vice-versa). In total, both parties exchange three tags to ensure that they have the same view of the protocol. This realizes the Authentication By Recognition as envisioned by our SEKA model, even in the presence of active adversaries. In case a parties accepts all incoming messages, it derives the session key via the KDF using the exchanged key k_{eph}, session nonce, local secret state as well as both identifiers as inputs.

Remember that in this phase, the adversary is able to perform active attacks such as injecting or modifying messages, as well as compromising the state, randomness or keys of honest parties. In case a preceding Bootstrap (or Key-Exchange) session was *clean*, and the adversary did not corrupt the pairwise *state*, the CK^{AM}_{PFS}-security of the underlying key exchange protocol guarantees that the adversary cannot distinguish the input state of the current

$\mathsf{Bootstrap}(pp, ID_i, ID_j, \mathcal{I}) \to st_{i,j}$		$\mathsf{Bootstrap}(pp, ID_j, ID_i, \mathcal{R}) \to st_{j,i}$
$s \leftarrow \{0,1\}^\tau$		
$k_{eph} \leftarrow \mathsf{KE.exchange}(pp, ID_i, ID_j, s, \mathcal{I})$	$\longleftrightarrow$	$k_{eph} \leftarrow \mathsf{KE.exchange}(pp, ID_j, ID_i, \bot, \mathcal{R})$
$st_{i,j_c} \leftarrow \mathsf{KDF}(k_{eph}, \tau, s, (0, ID_i, ID_j))$		$st_{j,i_c} \leftarrow \mathsf{KDF}(k_{eph}, \tau, s, (0, ID_i, ID_j))$
return $(st_{i,j_c}, \emptyset)$		**return** $(st_{j,i_c}, \emptyset)$

Figure 3: Bootstrap phase of the π_{SEKA} protocol.

$\mathsf{Key\text{-}Exchange}(pp, (st_{i,j_c}, st_{i,j_p}), ID_i, ID_j, \mathcal{I})$ $\to (k, (st_{i,j_c}', st_{i,j_p}'))$		$\mathsf{Key\text{-}Exchange}(pp, (st_{j,i_c}, st_{j,i_p}), ID_j, ID_i, \mathcal{R})$ $\to (k, (st_{j,i_c}', st_{j,i_p}'))$
$s \leftarrow \{0,1\}^\tau$		
$m_1, st_{eph}, \_ \leftarrow \mathsf{KE.handle}(pp, ID_i, ID_j, s, \mathcal{I}, \bot, \bot)$		
$t_1 \leftarrow \mathsf{Mac}(st_{i,j_c}, (ID_i, s, m_1))$	$\xrightarrow{ID_i, s, m_1, t_1}$	**abort if** $\forall st' \in st_{j,i_p} \cup \{st_{j,i_c}\}$: $\mathsf{Vf}(st', (ID_i, s, m_1), t_1) = 0$ **if** $\mathsf{Vf}(st_{j,i_c}, (ID_i, s, m_1), t_1) = 0 \wedge$ $\exists st' \in st_{j,i_p} : \mathsf{Vf}(st', (ID_i, s, m_1), t_1) = 1$: **persist** $st_{j,i} \leftarrow (st', \emptyset)$ **else** $st' \leftarrow st_{j,i_c}$ $m_2, \_, k_{eph} \leftarrow \mathsf{KE.handle}(pp, ID_j, ID_i, s, \mathcal{R}, \bot, m_1)$ $st_{j,i_c}', k \leftarrow \mathsf{KDF}(k_{eph}, 2 \cdot \tau, s, (st', ID_i, ID_j))$ **persist** $st_{j,i} \leftarrow (st', st_{j,i_p} \cup \{st_{j,i_c}'\})$
abort if $\mathsf{Vf}(st_{i,j_c}, (ID_j, s, m_2, m_1, ID_i), t_2) = 0$	$\xleftarrow{ID_i, s, m_2, t_2}$	$t_2 \leftarrow \mathsf{Mac}(st', (ID_j, s, m_2, m_1, ID_i))$
$\_, \_, k_{eph} \leftarrow \mathsf{KE.handle}(pp, ID_i, ID_j, s, \mathcal{R}, st_{eph}, m_2)$ $st_{i,j_c}', k \leftarrow \mathsf{KDF}(k_{eph}, 2 \cdot \tau, s, (st_{i,j_c}, ID_i, ID_j))$		
$t_3 \leftarrow \mathsf{Mac}(st_{i,j_c}, (ID_i, s, m_1, m_2, ID_j))$	$\xrightarrow{ID_i, s, t_3}$	
		abort if $\mathsf{Vf}(st', (ID_i, s, m_1, m_2, ID_j), t_3) = 0$
return $(k, (st_{i,j_c}', \emptyset))$		**return** $(k, (st_{j,i_c}', \emptyset))$

Figure 4: Stateful authenticated key exchange that implements the Key-Exchange phase, adapted from the signature-authenticated key exchange by [8] and the state transformation by [11].

Key-Exchange session from random. Thereby, using the same argument as Canetti and Krawczyk, one can prove that an *active* adversary cannot distinguish the session key of the later sessions from random.

Robustness. Our protocol must take extra care to not get out of sync, in case an adversary drops or alters messages between two honest parties. To this end, just as in the construction by [11], the Key-Exchange phase maintains a list st_p of *potential* states as well as the *current* state st_c. Thereby, if the last message of a Key-Exchange session is lost, the responder can *recover* the st_c used by the initiator from st_p by using the first Mac tag to detect which st_c the initiator uses, updating its st_c if necessary.

As soon as the responder computes the ephemeral session key (but before the session accepts), it adds the output state of the current session to st_p. If a session accepts, st_p is cleared and st_c is updated. Thereby, the responder has knowledge of all states that the initiator might have at the current time. Note that the current state st_c of both parties need not be the same at the time the initiator executes the next session, because the adversary might have modified or dropped the third protocol message, causing the session at the responder not to accept. In this case, verifying the first MAC tag in the next conversation with st_c fails in the responder. It can now iterate st_p, determining which of the contained states the initiator used for the session, and replace st_c accordingly.

3.2. Proof Sketch

In order to prove the security of the construction under the security model, the following needs to be proven:

Theorem 1. Let π_{SEKA} be the construction given in Section 3. Let KDF be a random oracle. Let MAC be a *deterministic* MAC scheme with $SNF^{CMA}_{\mathcal{A},\Pi}$-security. Let KE provide CK^{AM}_{PFS}-security and weak perfect forward secrecy. Then π_{SEKA} is *correct*, *robust* and *secure* according to Definitions 2 and 4.

The correctness of the protocol follows immediately from the correctness properties of the building blocks. Also, the robustness can be proven by the same argument that was used by [11]. Hence, this section only gives a short proof sketch for the security of the presented π_{SEKA} construction. More detailed proofs of correctness, robustness and security can be found in Appendix B.

Session-key secrecy will be proven using the sequences-of-games technique, requiring three major steps that combine arguments from two previous works [8, 11]: In the first step, using a reduction to the CK^{AM}_{PFS}-security of the underlying key exchange protocol, it is proven that the adversary cannot distinguish the output state of the *clean* session from random. In the second step, utilizing a reduction to the $SNF^{CMA}_{\mathcal{A},\Pi}$-security of the MAC scheme, it is proven that the adversary cannot violate authentication in Key-Exchange. Thereby, it can be shown that the adversary cannot query the session key it is supposed to guess using the *session-key* oracle. This step relies on the *determinism* of the MAC scheme, as the adversary has full control over random number generation outside the *clean* session. This step has essentially already been proven by Canetti and Krawczyk [8]. In the final step, it is argued that the freshness predicates leave no room for a trivial attack against the construction. Combining all steps, one can see that the adversary cannot break security of π_{SEKA} without breaking an underlying primitive, proving security of the scheme.

In the first major step (Game 2), it is proven that the adversary cannot predict the ephemeral key computed during the *clean* session. Due to the restrictions laid out in the freshness predicates, the adversary has no powers over the *clean* session that it does not have in the CK^{AM}_{PFS}-game against the underlying key exchange protocol. Remember that CK^{AM}_{PFS} was defined by [8] and considers a *passive* adversary without knowledge of random numbers used in the test session. As CK^{AM}_{PFS}-security is required for the underlying key exchange protocol, a simple reduction can show that if an adversary can distinguish the ephemeral key computed in the *clean* session from random, it can break the CK^{AM}_{PFS}-security of the key exchange, which was assumed to be infeasible. Thereby, it is known that the adversary cannot distinguish the ephemeral key of the *clean* session from random, implying that the same is true for the session key and output state of the session due to the random oracle assumption. Informally, this step captures both *bootstrapping* and *recovery* of the pairwise secret, relying on passivity of the adversary which was motivated by the characteristics of the light channel.

In the second major step (Game 5), the fact that the *test* session and the *clean* session are *not the same session* that was established in the first step is used. The goal of the second step is to prove that no session that computes the same session key as the test session without partially matching it exists. Thereby, according to the freshness conditions, the adversary is not allowed use the $session - key$ oracle to query the session key computed by the test session. In other words, the *authentication* provided by Key-Exchange is proven here. Because of the restrictions imposed on the adversary by the freshness conditions, proving this step is *very similar* (but not *formally equivalent* due to differences in the formal model) to proving the CK^{UM}_{PFS}-security of Key-Exchange, i.e., session-key secrecy in the presence of an *active* adversary *without knowledge of the long-term secrets used in the session*. Therefore, a proof similar to the one presented by Canetti and Krawczyk is used, accounting for the differences between the models and the fact that Mac tags rather than signatures are used [8]. Essentially, this step is proven by a reduction to the $SNF^{CMA}_{\mathcal{A},\Pi}$-security of the Mac scheme. Remember that in the $SNF^{CMA}_{\mathcal{A},\Pi}$-game, the adversary has *no control* over random number generation. As the adversary controls the random number generation during the *test* session, however, the *determinism* of the Mac scheme is crucial for this step.

The third step (which is broken up into Games 3, 4 and 6) is to argue that the adversary cannot use its oracles in order to retrieve the input state of the test session without violating the freshness conditions and losing the game. Thereby, this step only serves the purpose of showing that no *trivial* attacks are enabled by the security model. The step also shows that the list of potential states st_p does not enable any attacks that would otherwise be impossible, as it is cleared in the *clean* session. This step of the proof is similar to the security proof of the PCS transformation provided by Cohn-Gordon et al. [11].

The proof concludes in Game 7 by stating that the only way left for the adversary to win the game is to predict the session key computed in the test session without using the $session - key$ oracle and without knowledge of the input state. Since KDF is a random oracle, this is only possible with negligible probability. Thereby, the presented construction is secure in the SEKA model.

4. Implementation and Evaluation

This section describes a prototype implementation of π_{SEKA} and evaluates its impact on processing time, transmission delay, and throughput. Our implementation was done on the same software and hardware that is used by a real prototype system. It utilizes only cryptographic primitives from libcrypto, which are available for bare-metal use (e.g., in WolfSSL). The first experiment measures the average computational duration of the π_{SEKA} phases in isolation, comparing the performance of SEKA and the commonly used WPA key exchange protocols. We show that π_{SEKA} noticeably outperforms the 4-Way Handshake (WPA2-PSK), EAP-TLS (WPA-Enterprise) and Dragonfly (WPA3-SAE), while providing stronger security and a more convenient setup. We then investigate the impact of SEKA on network latency and throughput.

4.1. π_{SEKA} Implementation

The reference implementation libSEKA of π_{SEKA} contains the proposed Key-Exchange and Bootstrap phases, and relies on OpenSSL 3.0 for the implementation of all cryptographic primitives. For our implementation, we first need to instantiate the three generic building blocks from π_{SEKA}, which is done as follows:

- KDF: HKDF with SHA2-512
- KE: X25519 [5], i.e., ECDH
- MAC: AES-128-GMAC (deterministic version)

KDF is instantiated as HKDF [15] with SHA2-512, as this instantiation is efficient and widely available in cryptographic libraries. X25519 [5], i.e., ECDH on the curve25519 elliptic curve, is used as key exchange. It provides high performance, a 128-bit security level and resists all known side-channel attacks against ECDH protocols.

AES-128-GMAC as deterministic MAC. Our choice of AES-128-GMAC [28] as message authentication scheme

requires some justification. Remember that a *deterministic* Mac function was required. By default, GMAC is *probabilistic*. Thus, utilizing GMAC as message authentication scheme might appear odd, as deterministic message authentication schemes such as SHA512-HMAC could be utilized instead. The reason this proposal uses GMAC is as follows. π_{SEKA} is expected to be implemented in conjunction with authenticated encryption and message authentication. Furthermore, authenticated encryption with GCM and message authentication with GMAC are often combined into a single functional unit in hardware implementations, reducing chip complexity and thus cost. In the interest of not requiring a second, functionally equivalent, building block instantiation for π_{SEKA} and thus additional implementation complexity, the proposal reuses GMAC for message authentication in π_{SEKA}.

To this end, a deterministic version of GMAC needs to be constructed. The key material of GMAC consists of a static part (the actual key) and a per-message nonce (the *initialization vector* or IV). With the exception of nonce selection, the implementation of GMAC is deterministic. NIST SP80038D, which standardizes GCM and GMAC, permits two constructions of the IV: *random selection* and *sequential selection* [14]. libSEKA relies on a sequential IV, making GMAC deterministic. In particular, the used IV consists of a pairwise-fixed *prefix* (a truncated SHA2-512-hash of the identities of both peers) and a 16-bit per-message *counter*. The counter implicitly becomes part of the computed *tag*, while the prefix is not transmitted explicitly, reducing overhead of the protocol. In case a counter is repeated for the same pairwise state, the corresponding message is dropped, following NIST guidelines [14]. It is assumed that each invocation of Key-Exchange is completed *before* the counter overflows. Should this ever happen, a new Bootstrap is required. As libSEKA assumes reliable transmission, the counter should not exceed 2 during honest protocol usage.

Authenticated Encryption. libSEKA also implements library functions for encryption and authentication of messages using the session key established by Key-Exchange. libSEKA provides message authentication with GMAC and authenticated encryption with GCM, again relying on the 128 bit AES cipher. Both GCM and GMAC use a sequential initialization vector as described for GMAC in Key-Exchange. This is permitted by SP80038D, as it does not lead to a security risk unless a combination of key and initialization vector is *repeated* [14]. Therefore, as soon as the counter overflows, the session key is *rotated*.

libSEKA can be used to directly integrate SEKA and authenticated encryption into the MAC layers of LiFi networks. In addition, both functionalities were used to build a layer-2 VPN, sekatunnel, allowing investigation of network performance in realistic load scenarios over any network, including loopback, Ethernet and LiFi networks which do not provide security themselves such as IEEE 802.15.13. The sekatunnel uses a custom link-layer encapsulation protocol, allowing it to keep utilizing the last session key computed by Key-Exchange for authentication or encryption while Key-Exchange is in progress. Also, sekatunnel uses *tap* devices for input/output of plain data frames and *AF_PACKET* sockets for input/output of encapsulated data frames.

4.2. Benchmark

The benchmark consisted of continuous runs of the SEKA phases over 30 seconds, measuring the duration for both parties. The parties were simulated in the same address space, exchanging messages via shared memory. 1000 warm-up runs were conducted. Six different CPUs were used[4]. The executable was pinned to one CPU thread using processor affinity. We used compiler flags *-O3 -march=native -mtune=native*.

Also, the three WPA key exchange protocols were benchmarked in the same setup: the 4-Way Handshake (WPA2-PSK), EAP-TLS (WPA-Enterprise) and Dragonfly (WPA3-SAE). To this end, a simple benchmark harness was built around iwd[5].

Hypothesis. SEKA should be highly efficient across all test platforms. Also, due to its simpler construction, SEKA is expected to outperform all WiFi key exchange protocols, possibly with the exception of the 4-Way Handshake. For all protocols, run-to-run-variance should be negligible.

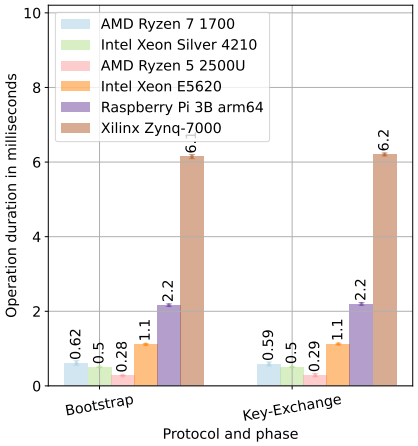

Figure 5: Average and standard deviation of execution times of SEKA protocol phases measured in the benchmark.

Results. Results of the benchmark can be found in Figure 5 and Figure 6. One can see that both π_{SEKA} phases have similar performance. Also, the x64 CPUs outperform the ARM CPUs, while all π_{SEKA} phases complete in less than ten milliseconds on all CPUs on average. Finally, Dragonfly and the EAP-TLS key exchanges are significantly slower than all π_{SEKA} protocols, while the 4-Way Handshake performs comparably to the π_{SEKA} protocols.

4. We used three workstation CPUs of different age (Intel Xeon Silver 4210, Intel Xeon E5620, AMD Ryzen 7 1700) with Ubuntu 20.04 LTS (Xeon 4210, Ryzen) and Debian 12 (Xeon E5620), a mobile CPU (AMD Ryzen 5 2500U) with Alpine Linux v3.17, a Raspberry Pi 3 model B with Alpine Linux v3.17 aarch64 and a dual-core Xilinx Zynq-7000 with Linux v5.15 armv7l configured with PREEMPT_RT.

5. https://web.archive.org/web/20230114085148/https://iwd.wiki.kernel.org/, date: 2023-03-14 16:40, utilized commit: *62301b7918160672009250a2db7e900c64df4440*. The benchmark of the 4-Way Handshake captures the key derivation from the PSK *"TestWPA2PSK"* as well as random number generation and the derivation of the Pairwise Transient Key in both parties. The benchmark of EAP-TLS relied on default configurations within iwd, specifically TLS 1.2 with 2048-bit RSA, AES-256-CBC, SHA-1 and ECDH on P-256. For Dragonfly, the passphrase *"secret123"* and the default curve (P-256) were selected.

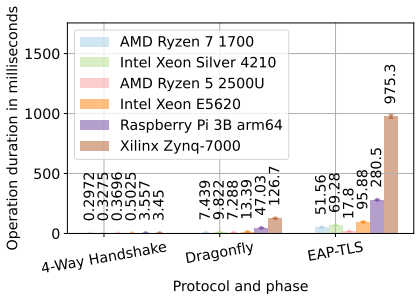

Figure 6: Average and standard deviation of execution times of WiFi key exchange protocols for reference.

We used the Linux perf(1) utility to record where CPU time was spent during the benchmarks. Perf records both time spent in userspace and in kernel space.

For the 4-Way Handshake, we have found that 98.34% of the computation time was spent in af_alg_sendmsg in the Linux kernel. This method's sole responsibility is to copy decryption buffers from userspace to kernel space for decryption (it does not perform any cryptographic operation). The reason for that is probably the overhead associated with copying buffers from user address space to kernel address space. Therefore, we must conclude that the overhead on the 4-Way Handshake is substantial. We assume that due to how fast the cryptographic operations are, they are not reflected in the perf record. All of the cryptographic operations required for SAE are implemented in userspace. Subtracting all method invocations that are not related to the cryptographic operations, we have found that 81.41% of the computation time of SAE was spent in cryptographic implementations. For EAP-TLS, we found that 88.26% of computation time was spent in the rsa_dec method (directly computing RSA signatures). Additional 0.15% of the computation time was spent computing SHA256 hashes. Additional 0.50% of computation time was spent verifying RSA signatures.

The perf records show that the SEKA implementation imposes only negligible overheads on the cryptographic primitives provided by OpenSSL. During the Bootstrap phase of SEKA, time was split almost evenly into generation of ephemeral states and computation of the ephemeral key, with only appx. 2% of computation time spent evaluating the key derivation function. During the Key-Exchange phase, appx. 2.5% of the overall computation time was additionally spent in the message authentication code generation and verification functions.

Conclusions. Analysis of the results shows that SEKA is suitable for use in LiFi networks. The high performance of the protocol enables both high data rates for individual devices and a large number of devices.

The significant performance differences between SEKA and Dragonfly or EAP-TLS have three probable causes. First, Dragonfly requires three modular exponentiations and EAP-TLS typically requires at least four modular exponentiations (key exchange and certificate/signature verification), while SEKA only needs two, per party. Second, exponentiations on Curve25519 are faster than on P-256 [5]. Curve25519 was not available in WPA at the time of writing. Finally, as [41] have shown, Dragonfly in particular is prone to timing-based side channels that allow an adversary to reveal the password from timing differences. The side channels require expensive mitigations, increasing the run time. To the best of our knowledge, the building blocks used for the π_{SEKA} do not suffer from this kind of side-channel vulnerability and require no additional mitigations for security in practice.

We expected the 4-Way Handshake to provide higher performance than measured. We conclude that the overhead on the 4-Way Handshake is substantial, and while the results reflect a real-world implementation of the protocol, they are not representative of the performance of the protocol itself. Still, comparable performance between π_{SEKA} and the 4-Way Handshake in practice is an impressive result. Recall that the 4-Way Handshake does not provide forward secrecy, and is prone to offline dictionary attacks, while π_{SEKA} provides far stronger security.

On the other hand, the benchmarks of EAP-TLS and SAE have an overhead of appx. 12-20%, which allows for fair comparison between the protocols and the SEKA protocols.

The results also show that the standard deviation for both π_{SEKA} and WPA is very small, with the exception of EAP-TLS. This matches the expectation. Using CPU affinity, we controlled for jitter introduced by scheduling and CPU boosting. Different random numbers and other input data should not affect the computation time in any way, as this would indicate possible timing-based side channel vulnerabilities. The slightly higher standard deviation of EAP-TLS can be explained by parsing and protocol overhead.

4.3. Impact on Delay & Throughput

As LiFi networks have been proposed that rely heavily on the physical protection instead of using cryptography, we further investigate the overall impact SEKA (and authenticated encryption) have on the latency and throughput of the link-layer protocol. Our analysis shows that SEKA in conjunction with AES-GCM is suitable for Time-Sensitive Networking (TSN), which is important for TSN-capable LiFi networks like IEEE 802.15.13. In terms of throughput, we demonstrate that the SEKA in conjunction with AES-GCM is capable of supporting a multi-gigabit-per-second LiFi network. All testbeds resemble the hardware and software used on real IEEE 802.15.13 prototype systems.

For space limitations we only summarize our study here, and refer to Appendix C for the detailed descriptions.

Impact on Delay. We investigated the impact of SEKA on network delays by deploying our sekatunnel VPN between two docker containers on the same host and measured round-trip communication delays.

The results of our experiment confirm that the performance impact SEKA imposes on link delay is negligible. In particular, SEKA and encryption impose a constant delay overhead of 10-20 microseconds to the link delay. At the same time, standard deviation and worst-case latencies of a reference measurement were virtually the same as in tests with SEKA.

We also measured the impact of SEKA on network delays on a proprietary IEEE 802.15.13 LiFi system. To this end, we integrated libSEKA into the MAC implementation. Due to limitations of the prototype system, we were

only able to test at a low data rate. Still, we were able to show that the latency impact of SEKA is negligible.

All in all, the results confirm that SEKA is suitable for usage in TSN-capable LiFi networks.

Impact on Throughput. We investigated the impact of SEKA on throughput with the same setup used for delay. Due to the throughput limitations of our LiFi test system at the time of writing, we were unable to perform a meaningfull throughput evaluation. As the wired network provides significantly higher throughput, we believe that our results are still representative of the real world, however.

We managed to achieve a throughput of up to four gigabits per second, which is the same as the throughput of the insecure reference measurement. Also, we managed to show that the throughput of SEKA increases linearly with the number of threads used for cryptographic operations until the limit of four gigabits per second is reached.

We concluded that hardware limitations caused by having to copy data in memory were the bottleneck in our test. Thereby, we are certain that SEKA is suitable for use in high-performance LiFi networks.

5. Discussion & Comparison

In this section we discuss the feasibility of our physical layer assumption, as well as what can be guaranteed if the assumption is not satisfied. We also compare the security and convenience of SEKA with other key exchange protocols that use a passively secure key exchange and discuss deployment considerations.

5.1. Passive Attack Limitation in LiFi

SEKA makes a strong assumption: *Passivity* of the adversary is required during at least one session (presumable the Bootstrap phase). Passivity of the adversary is generally considered a naïve assumption, since key exchange protocols are not usually restricted to usage on link-layer networks. In a network-, transport- or application-layer-protocol, devices are not able to control the path a message takes through the network. Thereby, it must be assumed that messages are relayed via compromised nodes, necessitating considering an active adversary.

However, in the envisioned use cases of SEKA, protocol messages can be exchanged directly via the optical medium without passing through other devices. Thus, in this scenario, the properties of the utilized link-layer network dictate whether a passive adversary is a reasonable assumption. As mentioned in the introduction, LiFi networks are believed to be confineable to a room. RF confinement using electromagnetic shielding is possible in the same fashion, albeit expensive and impractical. Light waves being absorbed by opaque obstacles makes shielding practical. Empirical research generally supports this assumption, with the limitation that parts of the message might be leaked to eavesdroppers [10]. LiFi is intended primarily for mobile machinery such as autonomous delivery robots in a larger factory hall, which can easily be moved into a more secure room for Bootstrap. For heavy stationary equipment, one can temporarily erect an opaque *shielding box* with an additional *repeater* OFE that shares a symmetric key with the network.

In order to perform a message-modification attack such as a MITM-attack in the same scenario, an adversary is not only required to ingest a decodable message into the room, but it also needs to completely cancel the original message. Otherwise, both parties can easily recognize the attack and abort the session. This makes the attack substantially more difficult than an eavesdropping attack. At the same time, assuming that Bootstrap is executed in a physically secured location, the adversary needs a high amount of physical access. However, if the adversary has this level of access, it is likely already able to achieve its goals *without compromising the LiFi network*. Thereby, we believe that the assumption is realistic in LiFi networks.

All in all, while we are comfortable assuming solely passive adversaries during the *short* Bootstrap (or a single Key-Exchange session), it is not realistic for the *entire* lifetime of the KE. Therein, attacks like Evil Twin/MAC spoofing are hard to prevent and SEKA must guarantee security against such active attacks.

Consequences of Violating the Assumption. In case the adversary is *not* passive during Bootstrap, SEKA can still bootstrap a secure state during a later session. This requires passivity of the adversary during an *arbitrary* future session. In other words, as soon as the adversary fails to corrupt messages in a *single* session, the compromise is healed and the adversary has no advantage in guessing future session keys. Due to device mobility, signal blockage and the directionality of LiFi channels, it is difficult for the adversary to continuously modify SEKA messages. If the adversary does not keep close proximity to the victim devices at all times, it is very likely that it will be unable to modify messages in a session at some point, which makes recovery of security likely.

A slightly modified (pessimistic) Bootstrap can provide security in scenarios in which passivity of the adversary cannot generally be assumed. To that end, Bootstrap can be implemented as key exchange protocol authenticated by a *short authenticated sequence*, e.g., using the construction published by [32]. Thereby, no passivity of the adversary in any session is required for the security of the protocol. Still, passivity of the adversary in a session can be utilized in order to heal prior compromise of device secrets or random number generation. Thereby, the pessimistic SEKA can still utilize the properties of the underlying channel for increased security.

5.2. Security-Usability Tradeoff

The design of SEKA provides a trade-off between security and usability that is more favorable for LiFi networks than what is provided by existing protocols. In this section, we summarize how SEKA provides higher security than existing configuration-free key exchange protocols with the same usability, higher efficiency than state-of-the-art authenticated key exchange protocols, stronger authentication than the password-authenticated key exchange protocols in the way they are commonly used in wireless networks and what implications these advantages have for practical use.

Security against Active Adversaries. LiFi is clearly not the only setting in which SEKA can be used. In fact, there are a number of configuration-free cryptographic protocols

that also exploit passive adversaries such as Bluetooth SSP (Secure Simple Pairing) [6], WPA3-OWE (Opportunistic Wireless Encryption) [17] and Push-Button WPS (Wireless Protected Setup) with WPA2-PSK or WPA3-SAE [1].

As was already discussed in Section 1.2, none of the aforementioned protocols can provide security against active adversaries if active attacks against the bootstrap phase are conducted. OWE does not even aim at providing *any* security against active adversaries [17]. In contrast, SEKA is fully secure against active attacks in the Key-Exchange phase (conditioned on passive attack security during Bootstrap), with strong Forward Secrecy and Post-Compromise Security as specified in our model. It can even heal from an active attack in the Bootstrap phase, if a single following session is not compromised.

Efficiency allows for periodic key updates. It was also demonstrated empirically that SEKA has significant performance benefits over EAP-TLS and Dragonfly. This allows execution of SEKA and Key-Exchange periodically, increasing the likelihood of recovering from any compromise while imposing a negligible overhead on delay and throughput. The efficiency gain also makes SEKA suitable for battery-powered or otherwise compute-constrained devices.

Authentication w/o Certificates. Finally, we would like to point out that SEKA provides stronger authentication than WPA-personal. In WPA2-PSK and WPA3-SAE, there is one passphrase *shared by all devices in the network*. Therefore, every device in the network can perform generic MITM attacks against all other devices in the network, breaking confidentialiy, integrity, authenticity and availability of communication. In particular, any device with knowledge of the network passphrase can impersonate any other device towards the network. In contrast, as SEKA negotiates a *pairwise* shared state that is only known to one initiator and one responder, one device in the network cannot impersonate a different device.

Still, key exchange protocols based on certificates such as EAP-TLS can provide stronger authentication. In addition to ensuring devices that they are communicating with the same device as in previous sessions, the PKI guarantees that the claimed identities are accurate. Also, this approach allows global revocation. As pointed out earlier, this comes at the cost of higher setup and performance overhead. Therefore, we believe that for almost all use cases of link-layer networks, the tradeoff between security and usability provided by SEKA is preferable.

5.3. Deployment Considerations

Finally, we discuss two practical considerations: topology changes and MAC address randomization.

Network Topology Changes. As was detailed earlier, a passive adversary during Bootstrap allows the elimination of out-of-band-channels in SEKA implementations. This makes SEKA highly suitable for embedded devices without I/O capabilities. A possible disadvantage of this approach is that once the network coordinator (responder) is replaced or a second network is created, *all* devices (initiators) need to be brought into the confined room separately in order to bootstrap a new shared secret. In large networks, this can cause significant costs. On the other hand, as lifecycles of especially LiFi network infrastructure devices such as coordinators (responders) are projected to be comparatively long [27], this problem occurs very infrequently. At the same time, an extension of SEKA can possibly utilize the secure channel between the devices (initiators) and the existing network coordinator (responder) in order to bootstrap a shared state with the new coordinator (responder) *outside* of the secure room.

Randomized MAC Addresses. A subtle but important difference between SEKA and, e.g., WPA lies in that SEKA expects the identities of two parties to remain constant. The security model also allows these identities to be known to the adversary, and the construction transmits them in the clear. At the time of writing, there is an ongoing effort in other link-layer key exchange protocols, such as the ones used in IEEE 802.11 [20] and 802.15.4 [21], to randomize the MAC addresses of mobile network devices. Thereby, it is made impossible for an eavesdropper to link two network sessions of the same device, improving privacy of the devices. Because of its reliance on static party identities, SEKA does not support similar privacy features. If privacy is a concern, initiators can change their identity regularly and complete a new Bootstrap. SEKA can also transmit the long-term identities of the parties encrypted with a key derived from the current shared state and associate long-term identities with randomized ephemeral session identities. Thereby, no additional Bootstrap invocations are required.

6. Conclusion

This paper introduced SEKA, a key exchange protocol designed for LiFi networks. SEKA improves upon two crucial weaknesses of existing wireless key exchange protocols: reliance on pre-installed keys and overhead in terms of performance and setup effort.

SEKA's *Bootstrapped Authentication* reduces device setup to a single button press. At the same time, *Authentication by Recognition* protects the network from impersonation and wardriving attacks. Finally, the protocol provides *Post-Compromise Security*, i.e., it can fully recover from complete compromise of a device's secrets.

All properties and required assumptions regarding the adversaries' behavior are precisely defined through a formal security model and a simple provably-secure construction is given. The assumptions on the physical security of the LiFi network are kept minimal and are required for short phases only. The implementation of our protocol shows that SEKA is significantly faster than state-of-the-art protocols for wireless key exchange and supports networking applications that are delay-critical or require multi-gigabit throughputs.

Still, improvements to SEKA are possible. For instance, investigating how to achieve a a stateless version of SEKA, which will be interesting when targeting devices with ephemeral filesystems. Further, we have already taken advantage of the SEKA model to replace signature-based authentication with MACs. Weaker versions version of the security model might even allow instantiations with only symmetric primitives, allowing for higher performance and post-quantum variants [3].

Acknowledgments

We thank the anonymous reviewers for their helpful feedback. The authors would also like to thank the Corporate Communications department at Fraunhofer HHI for their help in designing the illustrations for this publication.

Data Availability

The source code of libSEKA and sekatunnel, which is used for the benchmark of SEKA in section 4, was developed in the context of an industry project and is part of proprietary software. Therefore, at this time, it can unfortunately not be cleared for public release. However, we believe that the detailed description of the protocol provided in this paper makes it easy to implement SEKA from scratch.

The modified version of iwd is available at https://github.com/WorldofJARcraft/iwd.git.

References

[1] Wi-Fi Alliance. Wi-fi protected setup™ specification version 2.0.8. *Wi-Fi Protected Setup™ Specification*, 2020. URL https://www.wi-fi.org/system/files/Wi-Fi_Protected_Setup_Specification_v2.0.8.pdf. URL date: 2023-10-30 11:48.

[2] Mohamed Amine Arfaoui, Mohammad Dehghani Soltani, Iman Tavakkolnia, Ali Ghrayeb, Majid Safari, Chadi M Assi, and Harald Haas. Physical layer security for visible light communication systems: A survey. *IEEE Communications Surveys & Tutorials*, 22(3), 2020. doi: 10.1109/COMST.2020.2988615.

[3] Gildas Avoine, Sébastien Canard, and Loïc Ferreira. Symmetric-key authenticated key exchange (sake) with perfect forward secrecy. In *The Cryptographers' Track at the RSA Conference 2020*, CT-RSA 2020, Berlin, Heidelberg, 2020. Springer-Verlag. doi: 10.1007/978-3-030-40186-3_10.

[4] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *13th Annual International Cryptology Conference*, CRYPTO 1993, 1993. doi: 10.1007/3-540-48329-2_21.

[5] Daniel J Bernstein. Curve25519: new diffie-hellman speed records. In *International Workshop on Public Key Cryptography*, Berlin, Heidelberg, 2006. Springer-Verlag. doi: 10.1007/11745853_14.

[6] Bluetooth SIG. Bluetooth core specification version 5.4. *Bluetooth Core Specification*, 2023. URL https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=556599. URL date: 2023-10-30 11:48.

[7] Victor Boyko, Philip MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using diffie-hellman. In *19th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'00, Berlin, Heidelberg, 2000. Springer-Verlag. doi: 10.1007/3-540-45539-6_12.

[8] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT'01, Berlin, Heidelberg, 2001. Springer-Verlag. doi: 10.1007/3-540-44987-6_28.

[9] Sunghwan Cho, Gaojie Chen, Justin P Coon, and Pei Xiao. Challenges in physical layer security for visible light communication systems. *Network*, 2(1), 2022. doi: 10.3390/network2010004.

[10] Jiska Classen, Joe Chen, Daniel Steinmetzer, Matthias Hollick, and Edward W Knightly. The spy next door: Eavesdropping on high throughput visible light communications. In *2nd International Workshop on Visible Light Communications Systems*, VLCS'15:, 2015. doi: 10.1145/2801073.2801075.

[11] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. On post-compromise security. In *2016 IEEE 29th Computer Security Foundations Symposium*, CSF 2016, New York, 2016. Institute of Electrical and Electronics Engineers. doi: 10.1109/CSF.2016.19.

[12] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. *Journal of Cryptology*, 33(4), 2020. doi: 10.1007/s00145-020-09360-1.

[13] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6), 1976. doi: 10.1109/TIT.1976.1055638.

[14] Morris J Dworkin. Recommendation for block cipher modes of operation: Galois/counter mode (gcm) for confidentiality and authentication. Technical Report NIST Special Publication (SP) 800-38D, National Institute of Standards and Technology, Gaithersburg, 2007.

[15] H. Krawczyk. Cryptographic extraction and key derivation: The hkdf scheme. In *30th International Cryptology Conference*, CRYPTO 2010, Berlin, Heidelberg, 2010. Springer-Verlag. doi: 10.1007/978-3-642-14623-7_34.

[16] Keijo Haataja and Pekka Toivanen. Two practical man-in-the-middle attacks on bluetooth secure simple pairing and countermeasures. *IEEE Transactions on Wireless Communications*, 9(1), 2010. doi: 10.1109/TWC.2010.01.090935.

[17] Dan Harkins and Warren "Ace" Kumari. Opportunistic Wireless Encryption. RFC 8110, 2017.

[18] IEEE SA. Ieee draft standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 7: Light communications. *P802.11bb/D4.1, October 2022*, 2022. URL https://ieeexplore.ieee.org/servlet/opac?punumber=10063240. URL date: 2023-10-31 11:40.

[19] IEEE SA. Ieee draft standard for multi-gigabit per second optical wireless communications (owc), with ranges up to 200 meters, for both stationary and mobile devices. *P802.15.13/D10.0, November 2022*, 2022. URL https://ieeexplore.ieee.org/servlet/opac?punumber=10205959. URL date: 2023-10-31 11:40.

[20] IEEE SA. Standard for information technology–telecommunications and information exchange

between systems local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment: Operation with randomized and changing mac addresses. https://web.archive.org/web/20230712093518/https://standards.ieee.org/ieee/802.11bh/10525/, 2023. URL Date: 2023-10-31 08:44.

[21] IEEE SA. Ieee 802.15 wsn™ task group 4ac (tg4ac) amendment to ieee standard 802.15.4-2020. https://web.archive.org/web/20230712093447/https://www.ieee802.org/15/pub/TG4ac.html, 2023. URL Date: 2023-10-31 08:46.

[22] International Telecommunication Union. High-speed indoor visible light communication transceiver – system architecture, physical layer and data link layer specification. *Rec. ITU-T G.9991-2*, 2019. URL https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-G.9991-202104-I!Amd2!PDF-E&type=items. URL date: 2023-10-31 11:43.

[23] Christopher P Kohlios and Thaier Hayajneh. A comprehensive attack flow model and security analysis for wi-fi and wpa3. *Electronics*, 7(11), 2018. doi: 10.3390/electronics7110284.

[24] Hugo Krawczyk. Hmqv: A high-performance secure diffie-hellman protocol. In *25th International Cryptology Conference*, CRYPTO 2005, Berlin, Heidelberg, 2005. Springer-Verlag. doi: 10.1007/11535218_33.

[25] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In *International Conference on Provable Security*, ProvSec 2007, Berlin, Heidelberg, 2007. Springer-Verlag. doi: 10.1007/978-3-540-75670-5_1.

[26] Eduardo Novella Lorente, Carlo Meijer, and Roel Verdult. Scrutinizing wpa2 password generating algorithms in wireless routers. In *9th USENIX Workshop on Offensive Technologies*, WOOT 15, 2015. URL https://www.usenix.org/conference/woot15/workshop-program/presentation/lorente&sa=U. URL Date: 2023-10-30 11:47.

[27] Carmen Mas-Machuca, Madeleine Kaufmann, Maximilian Riegel, Dominic Schulz, Pieter Stobbelaar, Marcel Müller, and Daniel Behnke. Techno-economics of lifi compared to wi-fi in industrial iot applications. In *48th Annual Conference of the IEEE Industrial Electronics Society*, IECON 2022, New York, 2022. Institute of Electrical and Electronics Engineers. doi: 10.1109/IECON49645.2022.9968851.

[28] David A McGrew and John Viega. The security and performance of the galois/counter mode (gcm) of operation. In *International Conference on Cryptology in India*, Indocrypt 2004, Berlin, Heidelberg, 2004. Springer-Verlag. doi: 10.1007/978-3-540-30556-9_27.

[29] Ayman Mostafa and Lutz Lampe. Securing visible light communications via friendly jamming. In *2014 IEEE Globecom Workshops*, GC Wkshps, New York, 2014. Institute of Electrical and Electronics Engineers. doi: 10.1109/glocomw.2014.7063485.

[30] Ayman Mostafa and Lutz Lampe. Physical-layer security for miso visible light communication channels. *IEEE Journal on Selected Areas in Communications*, 33(9), 2015. doi: 10.1109/jsac.2015.2432513.

[31] Rohit Negi and Satashu Goel. Secret communication using artificial noise. In *2005 IEEE 62nd Vehicular Technology Conference*, VTC-2005-Fall, New York, 2005. Institute of Electrical and Electronics Engineers. doi: 10.1109/vetecf.2005.1558439.

[32] Sylvain Pasini and Serge Vaudenay. Sas-based authenticated key agreement. In *Public Key Cryptography: 9th International Workshop on Practice and Theory in Public Key Cryptography*, PKC 2006, Berlin, Heidelberg, 2006. Springer-Verlag. doi: 10.1007/11745853_26.

[33] Ge Shi, Yong Li, Wei Cheng, Xiang Gao, and Limeng Dong. Artificial-noise aided secure transmission over visible light communication system under coexistent passive and active eavesdroppers. *Optics Express*, 30(5), 2022. doi: 10.1364/oe.448860.

[34] Irina Stefan, Harald Burchardt, and Harald Haas. Area spectral efficiency performance comparison between vlc and rf femtocell networks. In *2013 IEEE International Conference on Communications*, ICC 2013, 2013. doi: 10.1109/ICC.2013.6655152.

[35] Chathura P. Suduwella, Yohani S. Ranasinghe, and Kasun de Zoysa. Visible light communication based authentication protocol designed for location based network connectivity. In *2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, 2017. doi: 10.1109/ICTER.2017.8257800.

[36] Chathura P Suduwella, Yohani S Ranasinghe, and Kasun De Zoysa. Visible light communication based authentication protocol designed for location based network connectivity. In *2017 Seventeenth International Conference on Advances in ICT for Emerging Regions*, ICTer 2017, New York, 2017. Institute of Electrical and Electronics Engineers. doi: 10.1109/ICTER.2017.8257800.

[37] Da-Zhi Sun and Li Sun. On secure simple pairing in bluetooth standard v5.0-part i: Authenticated link key security and its home automation and entertainment applications. *Sensors*, 19(5), 2019. doi: 10.3390/s19051158.

[38] Da-Zhi Sun, Yi Mu, and Willy Susilo. Man-in-the-middle attacks on secure simple pairing in bluetooth standard v5.0 and its countermeasure. *Personal and Ubiquitous Computing*, 22(1), 2018. doi: 10.1007/s00779-017-1081-6.

[39] A Lee Swindlehurst. Fixed sinr solutions for the mimo wiretap channel. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, New York, 2009. Institute of Electrical and Electronics Engineers. doi: 10.1109/icassp.2009.4960114.

[40] Achilleas Tsitroulis, Dimitris Lampoudis, and Emmanuel Tsekleves. Exposing wpa2 security protocol vulnerabilities. *International Journal of Information and Computer Security*, 6, 2014. doi: 10.1504/IJICS.2014.059797.

[41] Mathy Vanhoef and Eyal Ronen. Dragonblood: Analyzing the dragonfly handshake of wpa3 and eap-pwd. In *2020 IEEE Symposium on Security and Privacy*, SP 2020, New York, 2020. Institute of

Electrical and Electronics Engineers. doi: 10.1109/SP40000.2020.00031.

[42] Stefan Viehböck. Brute forcing wi-fi protected setup, 2011. URL https://web.archive.org/web/20220522173328/https://sec-consult.com/fileadmin/user_upload/sec-consult/Dynamisch/Blogartikel/Old_Blogposts/sec-consult-kcodes-netsub-viehboeck.pdf. URL date: 2023-03-29 11:32.

[43] Chun Wang, Steve T K Jan, Hang Hu, Douglas Bossart, and Gang Wang. The next domino to fall: Empirical analysis of user passwords across online services. In *Eighth ACM Conference on Data and Application Security and Privacy*, DBSec'18, New York, 2018. Association for Computing Machinery. doi: 10.1145/3176258.3176332.

[44] Yiru Wang and Lin Zhang. High security orthogonal factorized channel scrambling scheme with location information embedded for mimo-based vlc system. In *2017 IEEE 85th Vehicular Technology Conference*, VTC-2017-Spring, New York, 2017. Institute of Electrical and Electronics Engineers. doi: 10.1109/VTCSpring.2017.8108583.

[45] Liang Yin and Harald Haas. Physical-layer security in multiuser visible light communication networks. *IEEE Journal on Selected Areas in Communications*, 36(1), 2017. doi: 10.1109/jsac.2017.2774429.

[46] Tatu Ylonen and Chris M Lonvick. The secure shell (ssh) transport layer protocol. RFC 4253, 2006.

[47] Junqing Zhang, Trung Q Duong, Alan Marshall, and Roger Woods. Key generation from wireless channels: A review. *IEEE Access*, 4, 2016. doi: 10.1109/ACCESS.2016.2521718.

[48] Moshe Zviran and William J Haga. Password security: An empirical study. *Journal of Management Information Systems*, 15, 1999. doi: 10.1080/07421222.1999.11518226.

A. Definitions and Security Requirements of Building Blocks

In the following subsections, the abstract cryptographic building blocks required to implement a SEKA are discussed in more detail. For each building block, a formal definition and the required security assumptions are given.

Key Derivation Function.

Definition 5 (Key Derivation Function). "A key derivation function is a function $\mathsf{KDF}(\sigma, l, r, c) \longrightarrow k$ that given keying material σ, length value l, optional salt r and optional context c outputs a string k of l bits." [15]

Within SEKA, the seed r is used for domain separation. In particular, it enforces independence of the probability distribution Σ of the keying material σ from KDF. The context is also used to bind the output to the identities and previous state of the parties.

The KDF is assumed to be a random oracle in the security proof.

Message Authentication Scheme. The SEKA also requires a message authentication scheme, which is defined as follows:

Definition 6 (Message Authentication (MAC) Scheme). Let $\mathcal{M}$ be a set of (potential) messages, $\mathcal{K}$ be a set of potential keys and $\mathcal{T}$ be a set of potential tags. A message authentication scheme for $\mathcal{M}$ is a triple of functions $\Pi = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ such that:

- $\mathsf{KGen}(1^\tau) \longrightarrow k$ On input the security parameter τ, the PPT function KGen generates a key $k \in \mathcal{K}$.
- $\mathsf{Mac}(k, m) \longrightarrow t$ On input key k and message $m \in \mathcal{M}$, the PPT function Mac generates a tag $t \in \mathcal{T}$.
- $\mathsf{Vf}(k, m, t) \longrightarrow b$ On input key k, message m and tag t, the deterministic Vf function returns $b \in \{0, 1\}$. If and only if $b = 1$, t is *valid* for k and m.

The message authentication scheme is *correct* if $\forall m \in \mathcal{M} : \forall k \in \mathcal{K} : \mathsf{Vf}(k, m, \mathsf{Mac}(k, m)) = 1$.

$SNF^{CMA}_{\mathcal{A},\Pi}$-security is required from the message authentication scheme:

Definition 7 (Strongly unforgeable MAC scheme). Let $\mathcal{A}$ be a PPT adversary. Define a $\mathsf{Mac}(k, \cdot) \longrightarrow t$ oracle that for each input message m_i returns $t_i \leftarrow \mathsf{Mac}(k, m_i)$ and logs both m_i and t_i. Define the $SNF^{CMA}_{\mathcal{A},\Pi}$-security game as follows:

$$k \leftarrow \mathsf{KGen}(1^\tau)$$
$$(m^*, t^*) \leftarrow \mathcal{A}^{\mathsf{Mac}(k,\cdot)}(1^\tau)$$

return 1 **if** $\mathsf{Vf}(k, m^*, t^*) = 1 \wedge \forall i : (m^*, t^*) \neq (m_i, t_i)$

return 0 **otherwise**

A message authentication scheme $\Pi = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ is strongly unforgeable, denoted as $SNF^{CMA}_{\mathcal{A},\Pi}$-secure, if $\forall \mathcal{A} : Pr[SNF^{CMA}_{\mathcal{A},\Pi}(\tau) = 1] \leq negl(\tau)$.

Furthermore, it is assumed that the Mac scheme continues to be $SNF^{CMA}_{\mathcal{A},\Pi}$-secure when the adversary has knowledge of or control over random number generation. This is enforced by requiring a *deterministic* message authentication scheme, like, e.g., GMAC with a sequential initialization vector. Obviously, if a scheme does not use random numbers, an adversary gains no advantage from controlling random number generation.

Key Exchange Protocol. The SEKA requires a key exchange protocol as defined by Canetti and Krawczyk [8]. In the CK model, a key exchange protocol is an n-party, message-driven protocol that constructs a session key. The key exchange protocol KE is modeled as follows:

- $\mathsf{init}(1^\tau) \longrightarrow pp$, the initialization function that given the security parameter τ provides public parameters pp. This models, e.g., the used group.
- $\mathsf{exchange}(pp, ID_i, ID_j, s, role) \longrightarrow k$, the invocation of the key exchange protocol which given public parameters pp, own identity ID_i, peer ID_j, session ID s and role $role \in \{\mathcal{I}, \mathcal{R}\}$ performs the key exchange with the peer and returns session key k. The parameter s is ignored if $role = \mathcal{R}$, and it is assumed that the session-id is conveyed within the key exchange protocol messages instead.
- $\mathsf{handle}(pp, ID_i, ID_j, s, role, st_{eph}, m_{in}) \longrightarrow (st'_{eph}, m_{out}, k)$, a message-by-message interface to the key exchange protocol that given public

parameters pp, identities ID_i and ID_j, role $role$, ephemeral session state st_{eph} and incoming message (or $\perp$) m_{in} generates an outgoing ephemeral state st'_{eph} or $\perp$, an outgoing message m_{out} or $\perp$ and on the final invocation a session key k. The parameter s is ignored in all invocations except the first invocation by the initiator, and assumed to be extracted from the key exchange messages instead.

The key exchange KE is required to be CK^{AM}_{PFS}-secure, i.e., secure against *passive* adversaries. The following additional properties are assumed:

1) The protocol exchanges two messages.
2) The protocol provides weak perfect forward secrecy, in other words, the adversary cannot distinguish session keys from random if it did not modify messages during the session, even if it compromised all long-term secrets of both parties [24]. This is trivially true for any protocol that does not assume the existence of a long-term secret, like the Diffie-Hellman protocol that will be used in the instantiation. For these protocols, weak perfect forward secrecy is directly implied by CK^{AM}_{PFS}-security, as the definitions are formally equivalent in this case.

B. Full Security Proof of π_{SEKA}

The full security proof of the proposed π_{SEKA} construction will be structured as follows. First, the correctness and post-network robustness properties of π_{SEKA} are proven. After that, a detailed proof of security is given.

B.1. Correctness Proof

The following needs to be proven:

Theorem 2. Two matching sessions output the same state $st'_{i,j} = st'_{j,i}$ and (if they correspond to Key-Exchange) the same session key $k_i = k_j$.

This can be proven as follows. Matching sessions have to be (1) accepted (2) between the partners in the roles that the involved parties expect (3) without modification of messages. The correctness of SEKA relies on the following invariant: the input state of both parties is always the same in a matching session. Using these conclusions, the following inductive argument proves the claim:

Induction Start. For the Bootstrap session, the correctness of the underlying key exchange protocol guarantees that the ephemeral key k_{eph} is the same. Hence, the inputs to KDF are the same and the output state is the same.

Induction Step. For any Key-Exchange session, assume that the input state is the same for both parties. This causes the output state and session key to be the same. In this case, the message authentication scheme is invoked with equivalent inputs for all tags. Because it is correct, all tags can be verified. This means that the sessions accept. Because the key exchange protocol is correct, the ephemeral key k_{eph} computed in both sessions is equal. Because of this, the inputs to KDF are equal, and the output state and session key are also equal.

B.2. Post-Network Robustness Proof

Formally, the post-network robustness of the Key-Exchange construction was proven by Cohn-Gordon et al. [11]. However, this proof assumes a pre-shared initial state that is computed from the public keys of both parties. Remember that in the post-network robustness game, the adversary is restricted to passivity during the Bootstrap phase between the parties. Thereby, the state output by Bootstrap is computed exactly the same as the initial state based on pre-shared public keys defined by [11]. Also, Bootstrap must be accepted at both parties before Key-Exchange. Hence, the same proof also covers the SEKA construction. Note that if a party is restricted to the initiator ($\mathcal{I}$) role, the implementation may omit st_{i,j_p} while still providing post-network robustness, as the potential states need never be updated.

B.3. Session-Key Secrecy Proof

This subsection focuses on formally proving that each game step given in section 3.2 does not change the probability of winning the modified game from negligible to non-negligible or vice versa. Let $\mathcal{A}$ be a PPT adversary. Let the *utilized state* of a session refer to the state st_{i,j_c} or st' that is used as a key for the message authentication scheme and as key derivation context. The event of $\mathcal{A}$ winning the game i or W_i is formalized as the event of the game outputting 1. Finally, it is assumed that $n_p \geq 2$ and $n_s \geq 2$. Otherwise, the adversary would not be able to create a valid *test* session and always lose the game.

Game 0. The original SEKA game.

Game 1. The same game as game 0, but the challenger randomly selects the indices $(p_c, p_{c'}, s_c, s_{c'})$ of the parties p_c, $p_{c'}$ and sessions within the parties s_c, $s_{c'}$. The challenger aborts the game if $(p_c, p_{c'}, s_c, s_{c'})$ do not correspond to the *clean* session c and its *partially matching* session c', respectively. In other words, the challenger guesses the parties and sessions that execute the *clean* session and its partially matching session and aborts if it guessed incorrectly. Remember that for each execution of the security game, n_p and n_s are fixed upper bounds for the numbers of parties and sessions, respectively. Hence, this step is a standard game hop based on a large failure event, and therefore $|Pr(W_1) - \frac{1}{2}| \leq negl(\tau) \iff |Pr(W_0) - \frac{1}{2}| \leq negl(\tau)$ □.

Game 2. The same game as game 1, but aborts if KDF is queried with key material k^{clean}_{eph}, which is the ephemeral key of the clean session. In this game, the challenger knows the clean session before it is created. As it simulates the computations of the honest parties itself, it always knows k^{clean}_{eph}. Thus, the challenger can compare all inputs made to the random oracle by $\mathcal{A}$ with the identified k^{clean}_{eph} and abort if it is ever input into the oracle. Because of the random oracle assumption, the challenger knows all inputs that the adversary queried from KDF [15].

Let $Pr(F)$ denote the event that the failure condition occurs. It needs to be proven that $Pr(F) \leq negl(\tau)$ in order to prove that $|Pr(W_2)| - |Pr(W_1)| \leq negl(\tau)$. Let ϵ_{KE} denote the advantage of an adversary winning the CK^{AM}_{PFS}-game. Using a reduction, it can be proven that $Pr(F) \leq \epsilon_{\mathsf{KE}} + negl(\tau)$. To this end, an adversary $\mathcal{A}'$ in the CK^{AM}_{PFS}-game can be constructed from an adversary $\mathcal{A}$ in game 2 as follows. For all sessions except the clean session, $\mathcal{A}'$ simulates all oracles like the challenger would. In the clean session, $\mathcal{A}'$ uses its oracles to retrieve m_1 and m_2 of the underlying key exchange protocol. Remember that the clean session c is the *last* session that fulfills the requirements laid out in the freshness attributes, i.e., its partially matching session c' is created *before* c. In this game, the challenger knows which session will be the partially matching session to the *clean* session and which session will be the *clean* session. By extension, $\mathcal{A}'$ can use the same approach of guessing these sessions and resort to randomly generating the output bit if it selected the wrong sessions. Thereby, $\mathcal{A}'$ can ingest m_1 into c' and m_2 into c, replacing the honestly generated messages of *only* this conversation with messages from the CK^{AM}_{PFS}-game in cases in which it guessed correctly. It then queries the *test-session* oracle in the CK^{AM}_{PFS}-game for this session, using the returned value as k^{clean}_{eph}, and performs the remaining computations in the protocol honestly. $\mathcal{A}'$ logs all KDF inputs. If $\mathcal{A}$ ever queries KDF with k^{clean}_{eph}, $\mathcal{A}'$ outputs 1. Else, it outputs 0.

$\mathcal{A}'$ has a polynomial run time if $\mathcal{A}$ has a polynomial run time. Also, in case $\mathcal{A}'$ does not abort, game 2 is simulated perfectly for $\mathcal{A}$ until the invocation of the random oracle that $\mathcal{A}'$ is waiting for. Because of the freshness requirements for the clean session, $\mathcal{A}$ cannot query the randomness of the clean session or modify messages without losing the game. Hence, only valid oracle queries in the CK^{AM}_{PFS}-game are made. The session key k and output state $st_{i,j}$ of the clean session are computed incorrectly if a truly random k^{clean}_{eph} is used instead of the real ephemeral key. However, due to the random oracle assumption, these outputs do not give the adversary any advantage in guessing k^{clean}_{eph}.

$Pr(F) \leq negl(\tau)$ can be shown as follows for $\mathcal{A}'$. Ignore executions in which $\mathcal{A}'$ did not guess the correct sessions for c and c', as in these cases, $\mathcal{A}'$ has a probability of exactly $\frac{1}{2}$ of winning the CK^{AM}_{PFS}-game. Consider the following cases in executions where $\mathcal{A}'$ guessed correctly:

1) Assume that a random key instead of the real k^{clean}_{eph} is returned. k^{clean}_{eph} is not used anywhere in the protocol except as an input for KDF. Hence, $\mathcal{A}$ has only a negligible chance of guessing k^{clean}_{eph}. This means that the key is only queried in the random oracle with probability $negl(\tau)$. Thereby, with probability $1 - negl(\tau)$, $\mathcal{A}'$ wins the CK^{AM}_{PFS}-game in this case.
2) Assume that the real k^{clean}_{eph} is returned. If k^{clean}_{eph} is queried in KDF by $\mathcal{A}$, then $\mathcal{A}'$ returns 1 and wins the game. The likelihood of $\mathcal{A}'$ winning the game in this case is exactly $Pr(F)$.

Overall, the likelihood of $\mathcal{A}'$ winning the CK^{AM}_{PFS}-game is $\frac{1+\frac{1}{(n_p \cdot n_s)^2} \cdot (Pr(F) - negl(\tau))}{2}$. Since the underlying key exchange protocol is CK^{AM}_{PFS}-secure, $Pr(F)$ must be negligible, proving $|Pr(W_2) - Pr(W_1)| \leq negl(\tau)$ $\square$.

Also, one can conclude that if the test session and the clean session are the same session, the likelihood of the adversary winning the game is negligible. Due to the freshness conditions, the adversary cannot query the session key of this session. Since k^{clean}_{eph} is never queried in the random oracle, $\mathcal{A}$ does not learn the session key from the oracle. Due to the random oracle assumption, there is no other way of learning the session key, which means that the advantage of $\mathcal{A}$ winning the game is negligible. Thereby, in the following steps, it can be assumed that the test session and the clean session are *not* the same.

Game 3. The same game as game 2, but aborts if the state $st_{i,j}$ utilized in any session between the *clean* and including the *test* session and between the partners of the test session is returned from the *corrupt* oracle. The likelihood of this event is negligible. This can be concluded from the freshness conditions in the SEKA game as follows.

By the protocol specification, as soon as the clean session and its partially matching session are accepted, the list of potential states st_{i,j_p} is cleared. That means that in the session following the clean session, the state $st_{i,j_c} = st_{j,i_c}$ computed in the clean session must be used. Because of the freshness conditions, the adversary cannot query the state utilized in any session between the clean and the test session and between the partners of the test session using the *corrupt* oracle. If the test session has a partially matching session, after the test session and any partially matching session have *accepted*, st_{i,j_p} is cleared in both parties and st_{i,j_c} is updated. Hence, the states utilized in any previous sessions are no longer contained in the output of the *corrupt* oracle. Also, if the test session has no partially matching session or it is not accepted, the adversary is prohibited from using the *corrupt* oracle against the party which did not clear its states indefinitely. The test session must have been accepted, which means that the states at its party are always cleared. Hence, all previous states cannot be learned from a *corrupt* query issued after the test session ends. Usage of the *corrupt* oracle between the clean and the test session is directly forbidden by the freshness conditions. Finally, because of the random oracle assumption, the inclusion of the partners of the session in the context argument and the chaining of the random oracle, one of the desired states is only repeated in an unrelated session with a negligible probability. This proves that $|Pr(W_3) - Pr(W_2)| \leq negl(\tau)$ $\square$.

Game 4. The same game as game 3, but aborts if the adversary queries KDF in a way such that the state st_{i,j_c} utilized in any session between the *clean* session and including the *test* session between the partners of the test session is returned. As was discussed in game 3, the challenger can trace the input of KDF generated by $\mathcal{A}$ using the random oracle, which means that the challenger can detect that the event occurred.

In this game, $\mathcal{A}$ is not allowed to query KDF with k^{clean}_{eph}. The output state $st_{i,j_c}{}^{clean}$ of the *clean* session

is the output of a random oracle whose input includes k_{eph}^{clean}. Thus, it cannot compute the output state ${st_{i,j_c}}^{clean}$ of the clean session directly. Hence, the adversary can only compute ${st_{i,j_c}}^{clean}$ with a negligible probability. Likewise, the states utilized in all sessions until and including the *test* session depend on the input state, which in turn cannot be computed without knowledge of ${st_{i,j_c}}^{clean}$.

Also, there is no other way that $\mathcal{A}$ can use to learn the state utilized in the relevant sessions. Let st_{i,j_c} be the state utilized in any session started after the *clean* up to including the *test* session between the partners of the test session. In this game, the adversary is not allowed to retrieve any of these input states using the *corrupt* oracle. Because of the random oracle assumption, all st_{i,j_c} can only be computed from an output state or session key with negligible probability. Furthermore, the message authentication scheme could not reach $SNF_{\mathcal{A},\Pi}^{CMA}$-security if the adversary was able to compute an st_{i,j_c} from the authentication tags. Finally, remember that this state may also be exchanged with a state from the list of potential states st_{i,j_p} during execution of the session. All states in st_{i,j_p} are cleared as soon as the *clean* session accepts. Hence, no state in st_{i,j_p} can be computed without knowledge of ${st_{i,j_c}}^{clean}$. That means that the adversary has only a negligible probability of guessing any potential state in st_{i,j_p}. In conclusion, $\mathcal{A}$ has only a negligible probability of guessing any input st_{i,j_c}, which means it cannot compute the output state using the random oracle. This proves that $|Pr(W_4) - Pr(W_3)| \leq negl(\tau)$ $\square$.

Game 5. The same game as game 4, but it aborts if any session that is not the test session and does not partially match the test session accepts and outputs the same session key k. In order to prove that $|Pr(W_5) - Pr(W_4)| \leq negl(\tau)$, assume that the failure condition has a non-negligible likelihood. One can see that this leads to a contradiction using the following argument.

Let s identify an *accepted* session that is not the test session and does not partially match it, but outputs the same session key. Since it was assumed that the failure condition is true, s exists. Also, if the session key k output by s is the same as in the test session, s must compute the same ephemeral key k_{eph} and utilize the same state st_{i,j_c}. Otherwise, by the random oracle assumption, the output is only the same as in the test session with negligible probability. That implies (1) that s accepted, but (2) the first or second message sent or received by it were altered from the ones sent by the test session or (3) the messages were unaltered, but session states, roles or identities do not match. For (1) to be true, $\mathcal{A}$ must have input one or two messages into the *send* oracle, targeting s, depending on $role_s$.

The probability that (3) is true and s exists is negligible. The identities and utilized states must be the same, and the roles must be different, for both parties to generate the same input for KDF. If the input of KDF is different, the likelihood that the session key matches is negligible.

However, for (2) to be true, $\mathcal{A}$ must *forge* authentication tags for the input messages. Both roles verify an authentication tag that includes both messages in order to accept. As each authentication tag is computed on an unique input tuple, an authentication tag *within* the session cannot be replayed by $\mathcal{A}$. Also, as discussed in the earlier games, the state utilized in s was never used before, which means replaying an earlier tag is impossible even if it was computed for the same input tuple of identities and messages. Thereby, there are only two ways to accomplish the forgery: (2.1) forging an authentication tag *without* knowledge of the utilized state, or (2.2) knowledge of the utilized state.

The negligibility of $\mathcal{A}$ accomplishing (2.1) can be reduced to the $SNF_{\mathcal{A},\Pi}^{CMA}$-security of the message authentication scheme. To that end, one can construct an adversary $\mathcal{A}'$ in the $SNF_{\mathcal{A},\Pi}^{CMA}$-game against the underlying MAC scheme from $\mathcal{A}$ in the following way. $\mathcal{A}'$ simulates all oracles before the test session in the same way the challenger would. In the test session, $\mathcal{A}'$ uses its own $\mathsf{Mac}(k,\cdot)$ oracle to compute the authentication tags for the exchanged messages. Furthermore, $\mathcal{A}$ uses a random session key instead of the session key output by the test session for the $test-session$ oracle. Since it simulates all parties, $\mathcal{A}'$ can compare the inputs to the $send$ oracle with the messages it generated and detect when a message has been altered. In this case, $\mathcal{A}'$ terminates and presents the message and tag as forgery in the $SNF_{\mathcal{A},\Pi}^{CMA}$-game.

Obviously, as long as $\mathcal{A}$ has a polynomial runtime, $\mathcal{A}'$ also has a polynomial runtime. One can see that neither the tags nor the session key in the test session are computed honestly. However, in (2.2), it will be shown that $\mathcal{A}$ has no advantage over guessing the state utilized in the session. Thereby, $\mathcal{A}$ cannot distinguish honestly computed authentication tags from authentication tags computed by the Mac oracle. Also, $\mathcal{A}$ cannot compute the session key of the test session using the random oracle. Additionally, as a deterministic MAC was required, no bit of the random number generator is used for the computation of the MAC, which allows $\mathcal{A}'$ to simulate the $cr-create$ and $randomness$ oracles correctly. Hence, the game simulated by $\mathcal{A}'$ is indistinguishable from the true SEKA game as long as $\mathcal{A}$ has no knowledge of the state utilized in the test session. That means that if $\mathcal{A}$ has a non-negligible advantage in winning the SEKA security game, it must have a non-negligible likelihood of querying the session key of the test session using the $session-key$ oracle. The only remaining way for $\mathcal{A}$ to achieve this is to modify a message or tag generated in the test session, making the session in the involved parties non-matching. In this case, there are two outcomes:

1) The modified message and tag are invalid. In this case, the receiver session aborts, and $\mathcal{A}$ cannot query its session key. $\mathcal{A}$ has a likelihood of $\frac{1}{2} + negl(\tau)$ of winning the game, as it gains no advantage from the manipulation.
2) The modified message and tag are valid. In this case, the combination of message and tag is a valid forgery in the $SNF_{\mathcal{A},\Pi}^{CMA}$-game, and $\mathcal{A}'$ wins the game.

Hence, one can see that if the advantage of $\mathcal{A}$ winning the SEKA game over guessing is not negligible, then the likelihood of $\mathcal{A}'$ winning the $SNF_{\mathcal{A},\Pi}^{CMA}$-game against the MAC scheme is not negligible. Since the MAC scheme is $SNF_{\mathcal{A},\Pi}^{CMA}$-secure, the advantage of $\mathcal{A}$ winning the SEKA game must be negligible. Thereby, the likelihood

of (2.1) is non-negligible. In order to accomplish (2.2), the adversary needs to know $st_{i,j_c'}$ utilized in s. In this game, the adversary is not allowed to query the state using the *corrupt* oracle or compute it using KDF, and it was argued that no other way to learn the state exists in game 4.

Hence, s only exists with negligible probability. All in all, this proves that $Pr(F) \leq negl(\tau)$, which implies that $|Pr(W_5) - Pr(W_4)| \leq negl(\tau)\square$.

Game 6. The same game as game 5, but aborts if a query $\mathsf{KDF}(k_{eph}^{test}, \cdot, \cdot, (st_{i,j_c}, ID_i, ID_j))$ with the ephemeral key of the test session k_{eph}^{test}, the state utilized in the test session $st_{i,j_c'}$ and the identities of the partners of the test session ID_i, ID_j and any salt and output length is issued.

By using public information and the *randomness* oracle, $\mathcal{A}$ may learn k_{eph}^{test} and the identities. However, as discussed previously, the adversary has only a negligible probability of guessing $st_{i,j_c'}$. Hence, the likelihood of the failure condition is negligible. Thereby, $|Pr(W_6) - Pr(W_5)| \leq negl(\tau)$.

Game 7. The same game as game 6, but replaces the session key k output by the *test* session with a random $k \leftarrow_R \mathcal{K}$. $|Pr(W_7) - Pr(W_6)| \leq negl(\tau)$ can be argued as follows. Executions of the game in which a session outputs the same session key as the test session but does not partially match it were excluded in game 5. Furthermore, the adversary is prohibited from using the *session-key* oracle to query the session key from the test session and all partially matching sessions in the *test-fresh* predicate. The output state of the test session is statistically independent from the session key due to the random oracle assumption. Thus, the adversary does not gain any advantage from querying the output state using the *corrupt* oracle. Finally, the adversary is not allowed to query KDF with the input necessary to re-compute the session key. Because the output of KDF is truly random, $\mathcal{A}$ only has a negligible advantage in guessing the session key. All in all, as the adversary is unable to guess the session key with non-negligible probability, the changes between the games are undetectable to the adversary and one can see that $|Pr(W_7) - Pr(W_6)| \leq negl(\tau)$.

However, in game 7, the session key k is selected uniformly at random irregardless of the bit b that the challenger selects. Hence, the output of the *test-session* query for each choice of b is computationally indistinguishable. Thereby, one can see that $|Pr(W_7) - \frac{1}{2}| \leq negl(\tau)$ $\square$.

Conclusion. For every game step it was proven that if and only if the advantage of any adversary $\mathcal{A}$ winning the game over guessing is negligible, the advantage of any adversary winning the original game over guessing is negligible. Finally, it was proven that the probability of the adversary winning the last game is $\frac{1}{2} + negl(\tau)$. Thereby, $Pr(W_0) \leq \frac{1}{2} + negl(\tau)$, proving the security of the construction in the SEKA security model $\square$.

C. Full Evaluation of Impact on Network Delay and Throughput

This section first introduces an additional optimization of SEKA that we used to increase network throughput. This section then details the setups and results of the experiments that were conducted to evaluate the impact of SEKA on network delay and throughput. Also, the conclusions from the experiments are discussed in detail.

C.1. Optimization: Stretching the Session Key

As an optional optimization, a deterministic function $\mathsf{Key-Rotate}$ can use the key derivation function to stretch a session key computed by SEKA by a configurable factor n_k. Assuming the key derivation function is a random oracle (or in practice a strong hash function), an adversary without knowledge of the SEKA session key cannot distinguish any of the n_k session keys from random. Thereby, this optimization does not jeopardize security of the scheme. However, the optimization reduces the number of invocations of SEKA by n_k. As the key derivation function can be assumed to execute significantly faster than SEKA, this reduces the impact the protocol has on latency and throughput.

C.2. Impact on Delay

We have evaluated the overhead SEKA imposes on network delays in two different scenarios. First, we deployed sekatunnel on a (loopback) wired network. Due to the very low baseline latency of the network, we are able to more accurately measure the overhead imposed by SEKA.

We also evaluate SEKA on an IEEE 802.15.13 LiFi prototype system. To this end, we encapsulated the protocol messages in IEEE 802.15.13 vendor-specific frames and added custom MAC header fields for authenticated encryption. We then used the API of libSEKA to add SEKA for key exchange in conjunction with authenticated encryption on top of the existing MAC procedures. Due to technological limitations of the LiFi network at the time of writing, we were only able to test at the lowest data rate provided by IEEE 802.15.13. Because of this fact and the TDMA scheduling, the latency baseline is higher and noisier, making it more difficult to deduce the exact impact of SEKA. However, the evaluation suffices to show that SEKA is practical for use in LiFi networks.

C.2.1. Wired Setup. *Setup.* This experiment investigated the impact that SEKA and authenticated encryption have on the latency of a link-layer network. To this end, a setup was built around the same Ryzen 7 1700-workstation that SEKA was already evaluated on in Section 4.2. Two instances of the sekatunnel were encapsulated into a network namespace using two Docker containers that shared a network bridge. Alpine Linux version 3.17 was used for both containers. The sekatunnel was launched with between one and four threads per container, and $n_k = 1$. That is, after 2^{16} packets, a new $\mathsf{Key\text{-}Exchange}$ was invoked, and $\mathsf{Key-Rotate}$ only derived one key from each session key. Thereby, this test was designed to be a worst-case scenario. Both containers were connected using a virtual bridge, facilitating communication at the link layer, while preventing the kernel from forwarding from one tap device to the other tap device without the packet being sent through the sekatunnel application. That

Scenario	M	σ	min	max
Reference	0.01	0.004	0.004	0.41
Tunnel w/o Security (1)	0.017	0.003	0.01	0.354
Tunnel w/o Security (2)	0.013	0.003	0.008	0.444
Authentication (1)	0.022	0.005	0.014	0.449
Authentication (2)	0.017	0.003	0.011	0.386
Encryption (1)	0.022	0.005	0.014	0.442
Encryption (2)	0.016	0.003	0.01	0.392

TABLE 3: Mean M, standard deviation σ, minimum min and maximum one-way latency max in milliseconds by measurement scenario in the wired setting. Number of threads in parentheses.

is, each packet input to the tap device was encapsulated into a sekatunnel frame and encrypted or authenticated, depending on the configuration.

Before each test, 1000 packets per second were sent using ping for 10 seconds, serving as a warmup run. One of the containers ran an iperf 2.1.5 server process in its network namespace, and the other container ran an iperf client process in its network namespace. iperf was configured for UDP, sending 10000 packets of the default length (1470 bytes) per second into the tap interface and running for 100 seconds. Two additional reference measurements were conducted: a measurement over sekatunnel without security, and a measurement via the network bridge, bypassing the sekatunnel application. Thereby, at least 14 invocations of Key-Exchange were completed during the test duration if authentication or encryption were enabled. For the two reference measurements, Key-Exchange was not executed during the test.

Hypothesis. Because of the asynchronous computation of Key-Exchange and the Key − Rotate function, there should only be negligible differences between the measurements with authentication and encryption and the reference measurements.

Results. In Figure 7 and Table 3, the results of the test runs can be found. In Figure 7 one can see that both reference measurements and the encryption enabled scenario are prone to small spikes in latency that are not correlated with the 14 periodic invocations of the Key-Exchange phase during the test with encryption. Table 3 contains a full statistical evaluation of all measurement scenarios. The evaluation shows that the average latency slightly increases when encryption or authentication are enabled, while latency stability remains essentially the same. In particular, even the largest spikes in latency are smaller than the computation time of Key-Exchange on the test platform that was measured in Section 4.2.

Conclusions. Differences between sekatunnel and the reference measurements are negligible in a practical application. Especially, the reference measurements show that all measured latency spikes can be explained by noise in the measurement. The sekatunnel uses elevated scheduling priority in order to minimize jitter introduced by scheduling and static ARP entries in order to eliminate jitter introduced by address resolution. Still, no strict bounds of the latency can be expected. The slightly higher average latency when the sekatunnel was enabled can be fully explained by the encapsulation overhead. The computation time of encryption or authentication explains why the average latency increases again when sekatunnel is started in a secure configuration. The difference between the maximum latency of the sekatunnel when no security protocol is active and when security is active is negligible, as was expected. The difference between the reference measurements and the sekatunnel measurements can be explained with the package encapsulation and tap device overheads. All in all, SEKA in conjunction with AES-GCM is suitable for Time-Sensitive Networking (TSN), which is important for TSN-capable LiFi networks like IEEE 802.15.13.

C.2.2. LiFi Setup. *Setup.* A setup representative of a real LiFi network was used to measure the performance impact of SEKA on real networks. To this end, the HILIGHT IEEE 802.15.13 prototype network was started with a coordinator and one network member, with a workstation connected to the gigabit Ethernet backhaul ports of coordinator and member. The coordinator utilized an Intel Xeon W3690 and Ubuntu Linux 20.04 with Linux 5.4.0-137-generic and glibc 2.31, and the member used the Xilinx Zynq 7000 CPU, the Linux 4.9.0-xilinx kernel and uClibc 1.0.34. OpenSSL 3.0.7 was used on both devices. Due to limitations of the LiFi system, the IEEE 802.15.13 PM-PHY at its slowest clock rate (12.5 MHz or appx. nine megabits per second) was used for the measurement. At slow clock rates, due to the resulting high frame transmission times, extra round-trips have a higher impact on round-trip times than on higher data rates. Hence, the values measured in this experiment are representative of a worst-case scenario.

The coordinator used one thread for encryption and decryption, respectively, while the member ran decryption and encryption on the same thread. Static internet protocol addresses were assigned to the Ethernet devices of the workstations.

During the first test, the ping utility provided by Ubuntu was used to send ICMP echo requests through the LiFi network to the other client device and measure the round-trip time. A reference measurement during which the network security and key exchange protocols were disabled was also conducted. 100,000 probes were sent with 100 probes per second, adapting the scenario used in the wired network scenario to the lower data rate of the PM-PHY. Due to the used parameters, no Key-Exchange invocation is expected to be necessary during the test run. Hence, this test specifically evaluates the latency impact imposed by the network security protocol. Bootstrap and the first Key-Exchange were completed before the measurement started, and ten probes were sent before the start of the test as a warm-up run.

For the second test, iperf version 2.0.13 was started in UDP server and client modes on both workstations. The clocks of both workstations were synchronized using the network time protocol for accurate time measurement. After that, a uniform UDP flow with a data rate of one megabit per second was sent through the network, and the inter-packet delays and latencies were measured. This allowed testing uplink (from member to coordinator) and downlink (from coordinator to member) directions separately. The test was run for one hour and transmitted 750.000 frames with $n_k = 10$, necessitating at least one

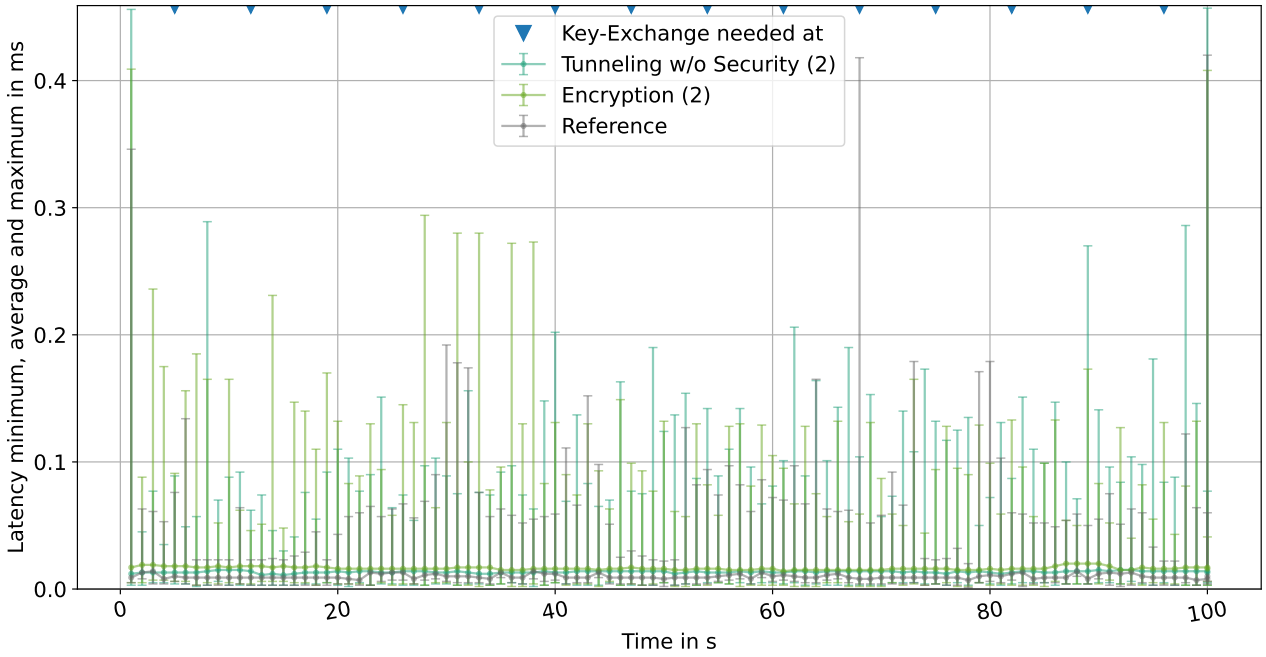

Figure 7: Average and range of one-way latency between initiator and responder container in the wired latency measurement setting over the test duration, aggregated for each one-second interval, comparing encryption enabled with two insecure reference configurations. Appx. start intervals of Key-Exchange invocations marked, revealing that latency spikes and Key-Exchange invocations are not correlated.

Scenario	M	σ	min	max
Authentication	3.92	0.42	2.6	14.60
Encryption	3.92	0.44	2.55	14.6
Reference	3.91	0.22	2.52	13.2

TABLE 4: Median M, standard deviation σ, minimum min and maximum round-trip time max in milliseconds by measurement scenario in the IEEE 802.15.13 setting.

invocation of Key-Exchange. Hence, this test is representative of long-term usage of the network and measures the effects of both network security and key exchange protocol. Due to the PM-PHY's lower data rate and the TDMA scheduling, if incoming data frames had to wait for a key exchange to complete, this would be evident in a latency spike. Since wired networks have faster round-trip times, the same scenario would be less obvious in a wired setting. Hence, this test setup was only used with the LiFi network.

Hypothesis. Because of the asynchronous computation of Key-Exchange and the Key – Rotate function, there should be only negligible differences between the SEKA measurements and the reference measurement.

Results. In figure 10 and table 4, the results of the first test can be found. Figure 8 and figure 9 contain the results of the second test.

Conclusions. All in all, the measurements for all three scenarios in all three tests are within the margin of error. Especially, one can see that the network security and key exchange protocol has no measurable impact on the high percentiles and maximum. As in the previous scenario, authentication and encryption slow the network down by less than 100 μs on average. This discrepancy can be explained by the computation time required and the overhead introduced by the upcall into libSEKA.

Furthermore, one can see that in the uplink scenario in particular, latencies even stabilize slightly when the network security protocol is enabled. In the reference measurement, the minimum-maximum-range of the latency oscillates by about 1.5 milliseconds. This is roughly the same as the TDMA slot size used by the scheduler in the given scenario. Hence, the oscillation can be explained by frames that arrive marginally too late and have to be scheduled in the next slot. The additional time used in the upcall likely reduces the likelihood of this scenario, stabilizing latencies.

Finally, note that the isolated latency spikes can be at least partially attributed to clock drift and synchronization between the measurement systems. Because of the high timing stability of the LiFi network, clock synchronization has a slight impact on the results.

C.3. Impact on Throughput

Setup. The final experiment in this section investigated the sustainable throughput. Using the same setup as in Section C.2, authenticated encryption was enabled for both communication directions and the number of threads launched in both sekatunnel processes was varied. Between 1 and 7 threads were launched, avoiding thrashing, and n_k was set to 10 in order to decrease computational overhead of the protocol. iperf was started in TCP mode with default settings and without bandwidth restriction, measuring the maximum sustained throughput through

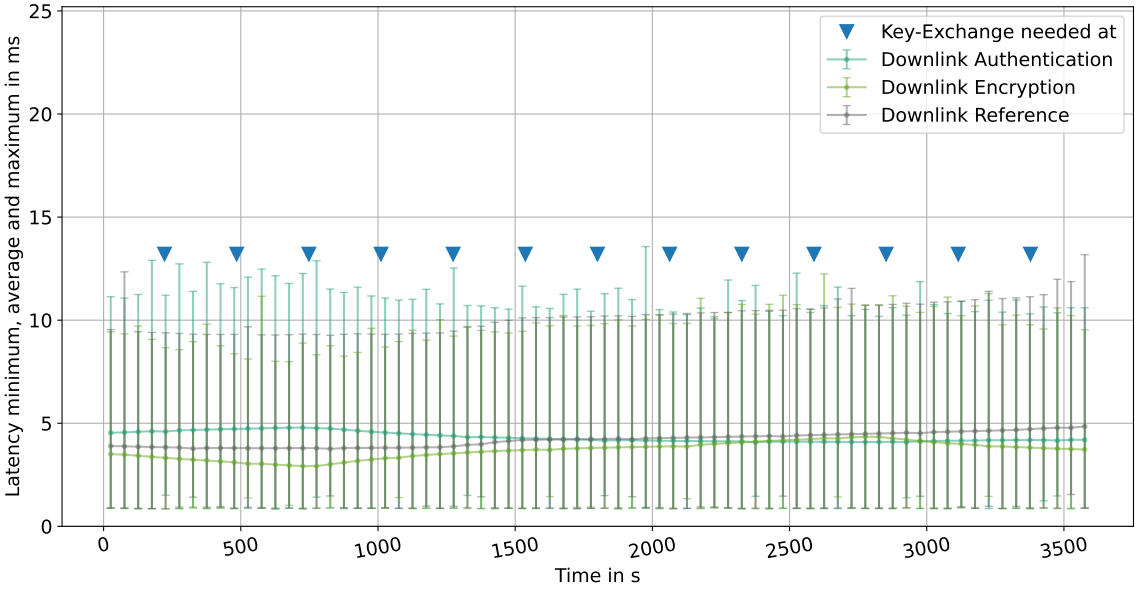

Figure 8: Average and range of one-way latency in the IEEE 802.15.13 setting, downlink measurement.

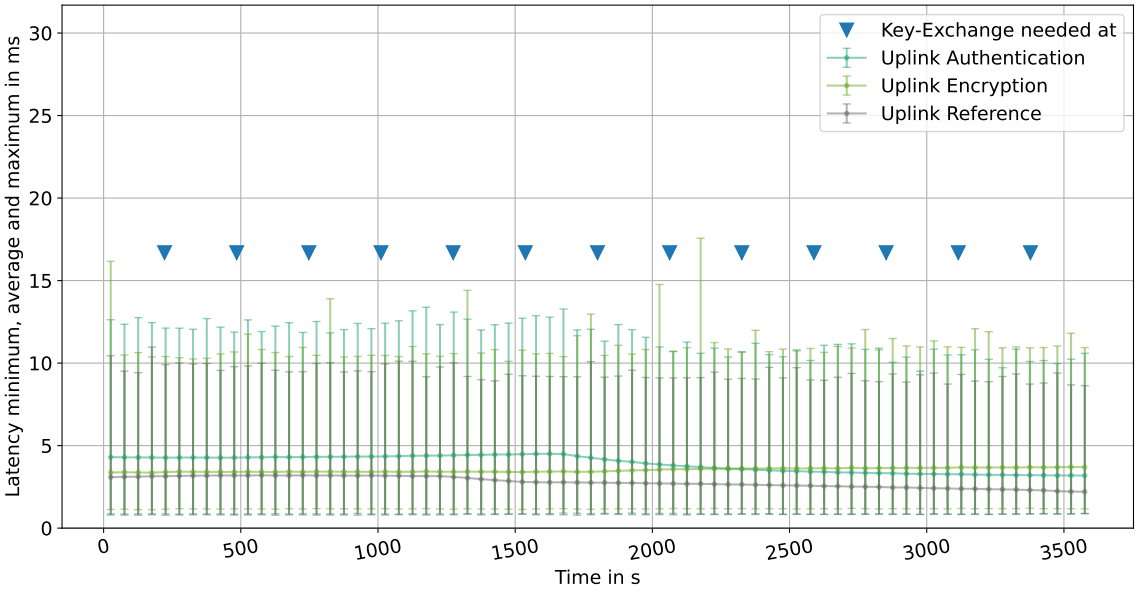

Figure 9: Average and range of one-way latency in the IEEE 802.15.13 setting, uplink measurement.

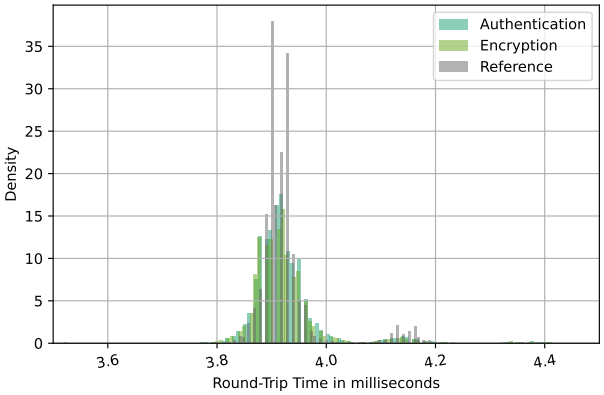

Figure 10: Histogram of round-trip times in milliseconds by scenario in the IEEE 802.15.13 setting.

the tunnel. As a reference, the throughput of sekatunnel without encryption or authentication was measured.

Hypothesis. The sekatunnel application provides a throughput above 1 Gigabit per second in all scenarios.

Results. Figure 11 illustrates the throughput in each scenario. The performance of the sekatunnel is identical both when encryption is enabled or disabled with 5 or more threads, while lower thread count reduces throughput.

Conclusions. The encapsulation in sekatunnel and the tap device seems to be the bottleneck in this scenario. This can be explained by the system call and context switch overhead. Also, as data needed to be copied between buffers in different address spaces for encapsulation, en-

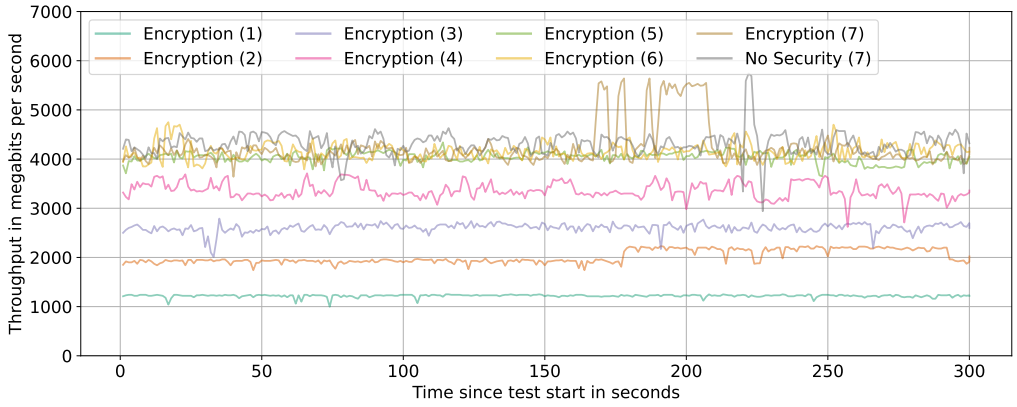

Figure 11: Receiver throughput over a loopback interface in megabits per second in different scenarios. Number of threads in parentheses.

cryption and decapsulation, the data rate of the used dual-channel DDR4-3000 memory modules might have limited the overall performance as well. However, one can also see that the sekatunnel scales with the number of threads launched, and that it is capable of providing multiple gigabits of throughput per second. This means that the SEKA in conjunction with AES-GCM is capable of supporting a multi-gigabit-per-second LiFi network.