




# Wireless-channel Key Exchange

Afonso Arriaga<sup>1</sup> , Petra Šala<sup>2</sup> , and Marjan Škrobot<sup>1</sup> 

<sup>1</sup> University of Luxembourg,  
`firstname.lastname@uni.lu`

<sup>2</sup> SES Techcom,  
`petra.sala@ses.com`

**Abstract.** Wireless-channel key exchange (WiKE) protocols that leverage Physical Layer Security (PLS) techniques could become an alternative solution for secure communication establishment, such as vehicular ad-hoc networks, wireless IoT networks, or cross-layer protocols.

In this paper, we provide a novel abstraction of WiKE protocols and present the first game-based security model for WiKE. Our result enables the analysis of security guarantees offered by these cross-layer protocols and allows the study of WiKE’s compositional aspects. Further, we address the potential problem of the slow-rate secret-key generation in WiKE due to inadequate environmental conditions that might render WiKE protocols impractical or undesirably slow. We explore a solution to such a problem by bootstrapping a low-entropy key coming as the output of WiKE using a Password Authenticated Key Exchange (PAKE). On top of the new security definition for WiKE and those which are well-established for PAKE, we build a compositional WiKE-then-PAKE model and define the minimum security requirements for the safe sequential composition of the two primitives in a black-box manner. Finally, we show the pitfalls of previous ad-hoc attempts to combine WiKE and PAKE.

**Keywords:** WiKE · wireless channel · key exchange · PAKE · physical layer security · cross-layer design

## 1 Introduction

Security and privacy in wireless communications has always been of foremost importance, but takes on a new dimension with the mass adoption of wireless-enabled devices propelled by the Internet of Things (IoT), wireless systems and other technologies such as as radio frequency identification (RFID) and vehicular ad-hoc networks (VANET). The traditional and most widely used approach to solving this problem is via key agreement protocols, which typically require legitimate parties to share a common secret key or password.

Protocols such as TLS, Kerberos, and Wi-Fi Protected Access are notable examples of widely deployed cryptographic solutions that incorporate Authenticated Key Exchange (AKE) or Password Authenticated Key Exchange (PAKE) mechanisms. These cryptographic primitives and the security guarantees arising

therefrom have been well-studied within standard security frameworks and under precisely formulated security definitions [14,6]. Within these security frameworks, an adversary is without exception modeled as a network adversary that has complete insight into the communication of honest participants. Security protocols following this paradigm are, in practice, deployed *above* the physical layer of the OSI model.

**Physical layer security.** An alternative security paradigm for enabling secure communication originates from the work of Wyner [37] and is implemented on the physical layer. The basic principle behind Physical Layer Security (PLS) arises from specific characteristics of the (wireless) communication channels. On a high level, the inherent random noise that affects communication channels can be leveraged to achieve information-theoretic security guarantees, albeit usually against an eavesdropping adversary. Since Wyner’s seminal work, various PLS techniques have been developed and are classified into two distinct groups [20]. The first group of techniques follows a keyless approach whereby (wireless) secret communication is directly enabled without relying on an encryption key. The second group relies on mechanisms that extract a sequence of random bits from the shared channel. The latter is the set of techniques we are interested in.

**Cross-layer design.** In recent years, there has been increasing research interest in hybrid security constructions [9,18] due to the very likely future threat of quantum adversaries to classical cryptographic primitives and protocols, but also as a result of the relative immaturity of existing quantum-secure schemes. In the domain of key exchange protocols, a hybrid approach involves a parallel execution of a classical key exchange protocol with a post-quantum key exchange protocol [9]. Outputs of both primitives can then be combined to obtain a master secret (to be used with symmetric-key primitives). Another potential solution to augment the security of communication systems and hedge against a motivated adversary is to consider a cross-layer security design [18]. In practice, key distribution problems are usually implemented above the physical layer. Moreover, the central purpose of the physical layer is usually only to provide an error-free link. However, one can also leverage the secrecy of wireless (and wired) networks and augment classical security measures by adopting physical layer security techniques. Assuming that involved end-point devices are secure and only their communication network is exposed, the use of the cross-layer (hybrid) approach would force an adversary to attack the targeted communication system in multiple domains simultaneously.

## 1.1 Our Contribution

We provide a detailed study of the Wireless-channel key exchange (WiKE), and our contributions can be placed into the following three categories:

**Wireless-channel key exchange model.** The design and security analysis of key exchange protocols has proved to be a difficult task. Even though many

WiKE protocols have been proposed during the last two decades [20], we are unaware of any attempt to describe a game-based or UC-based security definition for WiKE. In this paper, to address this gap, we propose a first, general, game-based security definition that captures the properties of WiKE. We base our security model on the Real-or-Random (RoR) variant [3] of the classical Bellare-Rogaway model for Authenticated Key Exchange (AKE) [7]. Our result provides a novel abstraction of WiKE protocols that allow them to be modelled within a standard provable security framework. We capture the difference (but also potential correlation) in communication between honest participants and an adversary. In contrast, in traditional key exchange protocols, all participating parties have the same view (i.e., noiseless transmission) of the network traffic.

**Composition with PAKE.** In this paper, we address the problem of a slow key generation rate due to inadequate environmental conditions that might cause the failure of WiKE in some circumstances. We explore a potential solution to such a problem by bootstrapping a low-entropy key from WiKE with a PAKE. We propose a generic solution building on top of our WiKE security model: we define a compositional WiKE-then-PAKE model following the techniques from [12] and [35]. Then, we prove that the sequential composition of any WiKE protocol secure in our RoR-WiKE model with any PAKE protocol secure in the standard RoR-PAKE model is also secure under the WiKE-then-PAKE security model. In this process, we observe that *forward secrecy* of RoR-PAKE is unnecessary for a safe sequential composition of the two primitives in a black-box manner.

**Insecurity of ad-hoc solutions.** The authors of [41] proposed a variant of PAKE called vPAKE, whose goal is to leverage the wireless fading channel in the physical layer to extract a common low-entropy key. Below, we show that their ad-hoc attempt to combine WiKE and PAKE has a circular argument in the security proof of the proposed PAKE protocol. Moreover, if deployed standalone, the proposed PAKE protocol allows testing if a client registers the same password with two different servers. Although the sequential combination of WiKE and vPAKE renders such an attack unfeasible because duplicate keys are unlikely to come out of WiKE, it's still noteworthy that the proposed protocol on its own is unsafe in most real-world scenarios. Interestingly, the attack that exploits this vulnerability is of practical significance and yet falls outside of the Real-or-Random game-based model.

## 1.2 Related Work

**Physical layer security.** In his seminal work, Wyner [37] considers an eavesdropping wire-tapper adversary with a degraded view of the communication channel between legitimate parties but assumes no pre-shared secret. Subsequently, Csiszár and Körner [16] generalized Wyner's result again in the noisy channel. Expanding on their work, Maurer looked at the problem of secret key generation from correlated information and noiseless public discussion [29]. He demonstrated that information-theoretic security is attainable if there exists only

a difference (and not necessarily an advantage) in the received signals between an eavesdropper and either of the legitimate parties. However, this result comes with a caveat: an additional, authenticated, error-free, public channel is needed. Later, Maurer and Wolf [28] analyzed a more difficult setting in which an adversary can actively participate in secret-key agreement protocol or certain parts of it. These works, among others, constitute the foundation for a relatively novel security research area of Physical Layer Security (PLS) and inspired a plethora of various schemes that are designed for different channel types, communication scenarios, and under various assumptions, [40,20].

**Password authenticated key exchange.** PAKE has been very heavily studied in the past 30 years. The idea of PAKE originates from the work of Bellare and Meritt [8]. The first formal models for analyzing PAKE emerged in the 2000s [6,11]. Bellare et al. [6] defined a game-based Find-then-Guess model (FtG) and showed that a provably secure PAKE protocol must provide two security properties: indistinguishability of the session key and authentication property. Abdalla et al. [3] extended their work and introduced a variant of the FtG model called Real-or-Random (RoR) that provides stronger security properties. In [32], Paterson and Stebila looked at the specificity of a one-time password scenario. The aspect of securely composing PAKE with other protocols was explored by Canetti et al. [14], where *Universally Composable* (UC) PAKE was first defined and the first UC secure construction was provided based on work from [25]. Their framework also captures possible correlations between passwords, which was not possible with previous game-based definitions. Over the years, many other PAKE protocols were proposed: for the latest survey, we refer to [21].

## 2 Preliminaries

In this section, we review two fundamental primitives that are used throughout this paper: Wireless-channel Key Exchange (WiKE), and Password Authenticated Key Exchange (PAKE).

### 2.1 Wireless-channel key exchange

The existence of a secure physical layer WiKE is dependent on several assumptions. The theoretical basis for WiKE assumes three physical phenomena that are observable in a typical multipath scattering environment [20]: 1) Spatial channel decorrelation; 2) Channel reciprocity; 3) Channel variation (randomness) that can exist in the temporal, spectral, and/or spatial domains. This means that the wireless channel between two communicants under real-world conditions produces a time-varying, random mapping between the transmitted and received signals. Importantly, this channel impulse response (mapping) is reciprocal, bound to communicants' location, and according to the Jakes uniform scattering model [22] decorrelates rapidly with the radio frequency (RF) half-wavelength distance due to the multipath fading phenomenon. Considering

a practical scenario where a wireless transmission occurs at 2.4 GHz, an eavesdropping adversary would have to be less than 6.25 cm away from either of the communicants to get meaningful information [26]. The aforementioned channel properties enable legitimate parties to first generate a “dirty” secret in presence of an eavesdropper that is later “purified”. To implement this in practice, most of the existing physical layer Wireless-channel Key Exchange (WiKE) schemes follow a 3-phase design commonly referred to as ‘advantage creation’, ‘information reconciliation’, and ‘privacy amplification’ [29,13,27].

**Phase I – Advantage creation.** The first phase starts with the successive probing of the wireless channel by the parties wishing to extract a secret key. Since the channel impulse response decorrelates in time, each probe can be seen as a fresh source of randomness<sup>3</sup> [39]. Unfortunately, this probing process is vulnerable to active attacks. Although specific physical layer authentication techniques exist [38], we cannot apply them directly to our problem, so we will assume an eavesdropping adversary in this phase of the protocol, as usually done in WiKE research. After the probing phase, communicating parties can transform correlated random measurements into correlated random bit strings through the process of quantization.

**Phase II – Information reconciliation.** After the first phase, the difference in the bit strings on the two sides is due to channel noise and interference, potential malicious participation of adversary, but also hardware limitations and vendor-specific implementation details [23]. This string mismatch is resolved using information reconciliation. As a result of this probabilistic, error-correction procedure legitimate partners end up with an identical random string  $S$ . This procedure typically assumes the existence of a noiseless, authenticated, public channel [34]. At this stage, the adversary may have partial information about  $S$ .

**Phase III – Privacy amplification.** This procedure solves the problem of leaked information during two previous phases and also removes correlations between subsequent bits in the string  $S$  that may occur because of a skewed estimate of the channel’s coherence time period. As a result, an insecure string  $S$  is compressed to a shorter string  $K$  that is almost uniformly distributed and outside the adversary’s knowledge. As with the information reconciliation procedure, the problem that privacy amplification solves is usually studied by assuming the existence of an error-free, authenticated channel. However, there exist protocols [28] that achieve privacy amplification without such assumption - security can be achieved in the presence of an adversary who possesses partial knowledge about the secret string  $S$ , but this knowledge must be limited [17].

---

<sup>3</sup> This is a simplification, as it assumes that each probe is done once during the channel’s coherence time-period. The problem is that it is usually difficult to estimate the exact coherence time period in the channel. However, this issue is typically addressed in the later WiKE phases.

**Authenticated channel.** In WiKE literature, similar to Quantum Key Distribution, it is typically assumed the existence of a secret setup established among WiKE protocol participants enabling an authenticated channel necessary for information reconciliation and privacy amplification. In practice, such a secret setup can be instantiated in multiple ways: using a pre-shared symmetric key, or by relying on a PKI, for instance. If one wants to achieve information-theoretic security, message authentication can be ensured using an unconditionally secure scheme such as Carter-Wegman MAC scheme [36]. However, since message authentication should only stay secure during the execution period of the WiKE, one can also resort to computationally-secure authentication [30].

**Comparing metrics.** WiKE schemes can be evaluated in terms of 3 important metrics [23]: 1) output entropy; 2) bit mismatch rate, and 3) secret key rate. The first two are self-explanatory, and the third metric quantifies the average number of secret bits extracted (per second) excluding bit losses due to information reconciliation and privacy amplification. Note that temporal channel variation, or in simple terms, movements of legitimate parties and other objects in the environment, are an important source of entropy and significantly contribute to the increase of the secret key rate.

**Security.** Adversarial threat models typically considered for WiKE assume only an eavesdropping adversary during the advantage creation (probing), as this phase is particularly sensitive to active adversaries. In contrast, the two subsequent phases may be achieved assuming an active adversary. Despite this limitation of WiKE, in contrast to more traditional key exchange approaches (e.g. Diffie-Hellman-based key exchange), WiKE's adversary is assumed to have an unbounded computational power and needs to be physically present and in close proximity to the protocol principals when WiKE is taking place. Therefore, a robust, well-designed, and thoroughly-implemented WiKE scheme should, in theory, only be affected by brute force attacks whose success depends on the length of the extracted key.

**Real-world deployment.** Although many physical layer security techniques and WiKEs have been proposed during the last two decades [20,33,31], we are only aware of WiKE being used in limited testbed environments [33].

## 2.2 Password Authenticated Key Exchange

Password Authenticated Key Exchange (PAKE) is a primitive that can be used over insecure networks to bootstrap weak pre-shared secrets (shared between two or more parties) into high-entropy secret keys. These low-entropy pre-shared secrets are in practice usually passwords, PINs, and passphrases, but they can also be partially secret strings. Although PAKE primitive is not a silver bullet for the key exchange problem, it can be very useful in certain scenarios. When compared with approaches using PKI, secret management in PAKE is simpler and more flexible. In the registration phase, protocol participants should secretly

exchange passwords (or bit strings of a certain amount of entropy) and fix public parameters that are known to everyone (including the adversary). It is important to note that PAKE protocols come in two flavours: balanced and augmented. A balanced PAKE protocol assumes that a secret shared among users is symmetric – it’s the same at both ends. Augmented (or asymmetric) PAKE is more suitable in a client-server setting where a server may wish to save a function of password to slow down the adversary in case of password file compromise. In this paper, we will be considering only balanced PAKEs.

**Security.** PAKE protocols must be free from offline dictionary attacks targeting users’ passwords. Online password guessing attempts must be recognized and limited to a small number per user account. In contrast to WiKE, PAKE offers security against fully active adversaries. However, adversarial interactions with honest parties using PAKE should provide the adversary with at most one password guess per user, and no other information should be leaked regarding the password used nor the resulting session keys. *Forward secrecy* guarantees that past communications remain confidential even in the event of a password compromise. This property is generally of great importance for standalone PAKE protocols, but as we show later in Section 4, it is unnecessary for the security of a black-box sequential composition of WiKE-then-PAKE. For precise security definitions for PAKE, we refer the to Appendix A, which describes the well-established Real-or-Random (RoR) model from [3].

**Real-world deployment.** In the past decade, we have seen a rise in popularity of large-scale deployments with PAKE. It is now used in electronic passports (ICAO Doc9303 standard), Wi-Fi Personal (WPA3), Apple’s iCloud, Thread protocol (IoT) to name a few [21].

### 3 Security Model for WiKE

Many Wireless-channel Key Exchange (WiKE) protocols have been proposed during the last 15 years [20]. However, we are unaware of any attempt to describe a game-based or UC-based security definition for WiKE. In this section, we intend to address this gap and propose a general game-based security definition for WiKE in the manner of Bellare-Rogaway (BR) Authenticated Key Exchange (AKE) models [6]. Within the model, the adversary interacts with participants via oracles with a well-defined interface. As typical for AKE protocols, the security property we are interested in is the indistinguishability of the session key in a multi-participant multi-instance setting.

#### 3.1 How to Model WiKE Security?

As explained in Section 2.1, almost all WiKE protocols consider an adversary with eavesdropping-only capabilities during the physical layer communication (i.e. advantage creation phase). The readings of an attacker obtained during

probing are correlated with those of legitimate parties but are also dependent on many factors (physical position, reading equipment, environment, etc). The two subsequent phases of WiKE (i.e. information reconciliation and privacy amplification) admit active adversaries and thus the interference of an attacker can be modelled per message flow, via Send queries, as usually done in game-based definitions of AKE and PAKE.

**Advantage creation modelling.** A number of environmental factors weigh in to determine the extracted channel features of the participants (legitimate or otherwise), such as the position of objects, whether the transmission takes place indoors or outdoors, noise, etc. Channel responses are similar at both ends of the same link (but not necessarily the same) and somewhat more decorrelated for an eavesdropping adversary who is more than half wavelength away. In the literature, the majority of PLS techniques are derived from the received signal strength indicator (RSSI) and channel state information (CSI), including phase and amplitude.

Similar to earlier works [29,13,27], we model the view of the adversary during the probing phase – with respect to legitimate parties – using a joint probability distribution. More formally, let  $X$ ,  $Y$ , and  $Z$  be discrete random variables with globally-known joint probability distribution  $\mathbb{D}_{X,Y,Z}$  and state space  $\mathbb{P}$ . The wireless channel behaviour is completely specified by  $\mathbb{D}_{X,Y,Z}$  that may be under partial control of an eavesdropping adversary. Let  $x$ ,  $y$ ,  $z$  be (possibly correlated) realizations of the random variables  $X$ ,  $Y$ , and  $Z$ , respectively. Here,  $x$  and  $y$  correspond to the view of legitimate participants and  $z$  corresponds to the view of the adversary measuring from a different position. We abstract away the channel quantization procedure by assuming that state space  $\mathbb{P}$  includes bitstrings of finite length.

### 3.2 WiKE Protocol

We represent the WiKE protocol as a pair of algorithms  $(WGen, W)$ .  $WGen$  is responsible for the generation of the secret(s) used to establish an authenticated link and of public parameters common to all principals.  $W$  defines how a WiKE protocol is executed internally by a protocol principal. In practice, WiKE protocol consists of three phases: advantage creation  $W.Phase1$ , information reconciliation  $W.Phase2$ , and privacy amplification  $W.Phase3$ . In our model, we treat these three phases as sub-algorithms of one monolithic algorithm  $W$ .

### 3.3 Real-or-Random Security Model for WiKE

We denote a game that represents the WiKE security model  $G^{wike}$ . In such a game, there exists a challenger  $C^{wike}$  whose job is to administer the security experiment and keep the appropriate secrets away from an adversary  $\mathcal{A}$  while doing so. We use  $\lambda$  to denote a security parameter.



**Protocol participants and execution.** In the two-party WiKE scenario, each node  $U$ , comes from a set of  $\mathbb{I}_{wike}$  that is a finite, nonempty set of identities in the form of bit strings. The protocol  $W$  is a PPT algorithm that describes the reaction of principals to the messages received, coming from both physical and upper network layers. In reality, each principal may run multiple executions of  $W$  with different nodes, thus in the model, each principal is allowed to run multiple instances by executing  $W$  in parallel. We denote  $U^i$  the  $i$ -th instance of principal  $U$ . In places where distinction matters, we will denote initiator instances  $T^i$  and responder instances  $R^j$ .

**Execution state of a principal instance.** Each principal's instance  $U^i$  holds an execution state that is updated as the protocol advances. The execution state contains all the necessary data for the protocol execution and is described as a tuple  $(U.setup, U^i.pid, U^i.sid, U^i.key, U^i.status, U^i.internal)$ , where:

- $U.setup$  might hold long-term secrets of  $U$ , either unique to  $U$  (such as a public/private key pair) or pre-shared secrets with other parties;
- $U^i.pid$  is the partner identifier of  $U^i$ , initially set to  $\perp$  and remains so until  $U^i$  starts running the protocol;
- $U^i.sid$  is the session identifier of  $U^i$  containing the full transcript of  $W.Phase2$  and  $W.Phase3$  of WiKE protocol;
- $U^i.key$  is the session key of  $U^i$ , and is set to  $\perp$  upon initialization and until the party instance  $U^i$  accepts;
- $U^i.status$  takes values from set  $\{running, accepted, terminated, rejected\}$ . It is set to *running* once an instance  $U^i$  is initiated, set to *accepted* once a running instance computes a session key  $U^i.key \neq \perp$ , set to *terminated* if the instance successfully terminates after accepting, and set to *rejected* if the instance could not compute a session key and aborted the protocol.
- $U^i.internal$  is an internal state reserved for any ephemeral state needed for the execution of WiKE protocol.

In an initialization phase of the execution state, which occurs before the execution of a protocol,  $WGen$  is run to generate the system's public parameters and long-term secrets. More specifically, before starting the game, the challenger  $C^{wike}$  generates long-term secrets via  $WGen$  such that every pair of parties  $(U, V)$  can establish an authenticated channel.

**Adversary.** When assessing the security of WiKE protocol  $W$ , we first need to define the adversarial capabilities. Our adversary  $\mathcal{A}$  runs in time  $t(\lambda)$ , which is possibly unbounded. In line with WiKE literature, we model  $\mathcal{A}$  with eavesdropping capabilities on the physical layer ( $W.Phase1$ ) and active capabilities on the upper network layers ( $W.Phase2$  and  $W.Phase3$ ).  $\mathcal{A}$  has access to principals' instances via certain oracles provided by  $C^{wike}$ . Upon receiving a query from  $\mathcal{A}$ ,  $C^{wike}$  parses it, forwards messages to corresponding instances, and sends their answer back to  $\mathcal{A}$ . Thus, while playing  $G^{wike}$ ,  $\mathcal{A}$  has the following set of queries:

- Execute**( $T^i, R^j$ ) This query models a honest run of  $W$  between initiator  $T^i$  and responder  $R^j$ . For the advantage creation phase,  $\mathcal{C}^{wike}$  samples three bitstrings  $x$ ,  $y$ , and  $z$  from the same finite set  $\mathbb{P}$  of size  $l \geq \lambda$  according to some joint probability distribution  $\mathbb{D}_{X,Y,Z}$ . While bitstring  $z$  is given to  $\mathcal{A}$ ,  $x$  and  $y$  are kept private. More precisely, value  $x$  is assigned as part of the internal state to  $T^i$  and  $y$  to  $R^j$ . The complete transcript related to both information reconciliation and privacy amplification phases is given to  $\mathcal{A}$ . As a result, instances compute the same key  $T^i.\text{key} = R^j.\text{key} \in \{0, 1\}^\lambda$  and  $T^i.\text{status} = R^j.\text{status} = \text{terminated}$ .
- Probe**( $T^i, R^j$ ) This query models an honest run of  $W.\text{Phase1}$  (advantage creation phase) between initiator  $T^i$  and responder  $R^j$ . In the same way, as for **Execute** query, three bitstrings  $x$ ,  $y$ , and  $z$  are sampled, and  $z$  is given to  $\mathcal{A}$ , while  $x$  and  $y$  are kept private. Thereby, the adversary cannot actively interfere during  $W.\text{Phase1}$ .
- Send**( $U^i, M$ ) This query models an active adversary for the phases  $W.\text{Phase2}$  and  $W.\text{Phase3}$ . As a result, a message  $M$  is sent to a principal instance  $U^i$  that responds to  $\mathcal{A}$  according to the protocol. Note that  $\mathcal{A}$  will be notified in case instance  $U^i$  accepts or terminates its execution.
- Reveal**( $U^i$ ) As a response to this query,  $\mathcal{A}$  receives the current value of the session key  $U^i.\text{key}$ .  $\mathcal{A}$  may ask this query only if  $U^i$  has successfully terminated (holding a session key) and a **Test** query has not been made to  $U^i$  or its partner instance. This query allows us to capture a potential leak of a session key as a result of its use in higher-level protocols. It ensures that in case some session key gets exposed, other session keys remain protected.
- Corrupt**( $U$ ) As a response to this query,  $\mathcal{A}$  receives the long-term secret value used by  $U$  to authenticate to its partner(s). Hence, this query models the security compromise of the authenticated channel. As we do not assume any particular instantiation of the authenticated channel, we leave this query agnostic to the type of trusted setup (e.g. a symmetric secret pre-shared pairwise, a public/private key pair per participant, etc.).
- Test**( $U^i$ ) At the beginning of  $G^{wike}$ , a hidden bit  $b$  is randomly selected by  $\mathcal{C}^{wike}$  and used for *all* **Test** queries. If  $b = 0$ ,  $\mathcal{A}$  receives  $U^i.\text{key}$  as an answer to the **Test**( $U^i$ ) query. Otherwise,  $\mathcal{A}$  receives a random string from the session key space  $\{0, 1\}^\lambda$ . In this case (i.e. when  $b = 1$ ),  $\mathcal{C}^{wike}$  must ensure that two *partnered* instances will respond with the same random value. It is important to note that only a *fresh* instance can be targeted with a **Test** query. This query is here to measure the indistinguishability of session keys.

The adversary is allowed to send multiple **Execute**, **Probe**, **Send**, **Reveal**, **Corrupt**, and **Test** queries to  $\mathcal{C}^{wike}$ . Note that the validity and format of each query are checked upon receipt. The session keys that are forwarded to  $\mathcal{A}$  in response to **Test** queries are either all real or all random.

**Game state.** In order to run a sound simulation, the challenger  $\mathcal{C}^{wike}$ , in addition to *execution states* of instances, maintains a *game state*. While  $\mathcal{C}^{wike}$

updates the execution state with the progression of the actual network interactions between  $\mathcal{A}$  with the instances running  $W$  on the lower level, the game state is updated with the progression of the security game  $G^{wike}$  on the higher level.  $\mathcal{C}^{wike}$  will flip the test bit  $b$  at the beginning of the game. All other flags – such as those related to freshness and partnering properties (see below), as well as those that track which instance is tested, corrupted, or revealed are maintained. From the adversary’s perspective, a pair of instances  $T^i$  and  $R^j$  come into being after either  $\text{Execute}(T^i, R^j)$  or  $\text{Probe}(T^i, R^j)$  query is asked.

**Partnering.** We say that instance  $T^i$  is a partner instance to  $R^j$  and vice versa if: (1)  $T$  is an initiator and  $R$  is a responder or vice versa, (2) both party instances hold same session identifiers  $\text{sid} = T^i.\text{sid} = R^j.\text{sid} \neq \perp$ , (3) both party instances hold appropriate partner identifiers  $T^i.\text{pid} = R$  and  $R^j.\text{pid} = T$ , (4) both party instances hold the same session keys  $T^i.\text{key} = R^j.\text{key}$ , and (5) no other instance has a non- $\perp$  session identity equal to  $\text{sid}$ .

**Freshness.** This property captures the idea that the adversary should not trivially know session keys being tested. First, an instance  $T^i$  and its partner instance  $R^j$  are made *fresh* after  $\text{Execute}(T^i, R^j)$  query is asked. Furthermore, an instance  $U^i$  (whether this is  $T^i$  or  $R^j$ ) that has accepted as a result of appropriate  $\text{Probe}$  and  $\text{Send}$  queries is *fresh* unless any of the following conditions hold: (1)  $\text{Reveal}(U^i)$  query was asked previously, or (2) if  $\text{Reveal}(V^j)$  query was asked previously where  $V^j$  is  $U^i$ ’s partner instance, or (3) if any participant  $Q$  was target of  $\text{Corrupt}(Q)$  query before  $U^i$  defined its key  $U^i.\text{key}$ , and a  $\text{Send}(U^i, M)$  query occurred.

**WiKE security.** Now we can formally define WiKE advantage of  $\mathcal{A}$  against  $W$ . Eventually,  $\mathcal{A}$  ends the game and outputs a bit  $b'$ . We say that  $\mathcal{A}$  wins the game if  $b = b'$ , where  $b$  is the hidden bit selected at the beginning of the protocol execution. We denote the probability of this event by  $\mathbb{P}[b = b']$ . The wike-advantage of  $\mathcal{A}$  in breaking  $W$  is defined as

$$Adv_W^{wike}(\mathcal{A}) \stackrel{\text{def}}{=} |2 \cdot \Pr[b = b'] - 1|. \quad (1)$$

Finally, we say that  $W$  is wike-secure (resp. *everlasting* wike-secure) if for every PPT (resp. *unbounded*) adversary  $\mathcal{A}$  it holds that

$$Adv_W^{wike}(\mathcal{A}) \leq \epsilon(\lambda), \quad (2)$$

where function  $\epsilon$  is negligible in the security parameter  $\lambda$  (that also defines the length of the session key output by  $W$ ).

This formula captures the idea that an adversary’s advantage in breaking a WiKE should only negligibly grow with the reduction in the length of session keys obtained as a result of WiKE protocol. In particular, a protocol secure in this model guarantees that generated session keys are indistinguishable from the uniform and independently sampled random keys.

**Remark 1.** In our model, we assume an eavesdropping adversary during the advantage creation phase due to its high sensitivity to active adversaries. We abstract away from different PLS techniques used in the advantage creation phase (probing, measurements, and quantization). We assume that honest instances and an eavesdropper each get a random (potentially correlated) bit string of a certain length sampled from some joint probability distribution. Such an approach allows us to capture various proposed PLS techniques.

**Remark 2.** Notice that in our model `Execute` query differs from the similar query in standard key exchange game-based models (e.g., [6]). More precisely, each protocol participant (including the adversary) has a distinct view of the result of the advantage creation phase due to variations in measurements (differences occurring due to location, hardware, timing, etc.). Thus, there is no single global transcript of the advantage creation phase. For this reason, the session id that uniquely names the WiKE session only includes messages from `W.Phase2` and `W.Phase3`. Otherwise, two partners would likely end up with distinct session ids as the first phase of `W.Phase1` runs over a noisy channel.

**Remark 3.** Although WiKE literature typically assumes information reconciliation and privacy amplification to occur over authenticated links (after advantage creation), we allow the adversary to send maliciously crafted messages via `Send` queries, which enables an adversary to try to defeat the message authentication. Moreover, such a choice enables analysis of various WiKEs, as there exist protocols for privacy amplification that achieve security against active adversaries without relying on authenticated links [28].

**Remark 4.** The spatial channel decorrelation assumption implies that any eavesdropper located more than one half-wavelength away from either initiator or responder experiences uncorrelated multipath fading<sup>4</sup>. More specifically, the value  $z$  that a distant eavesdropper receives is uncorrelated with values  $x$  and  $y$  obtained by honest parties. At the same time, due to channel reciprocity property,  $x$  and  $y$  values should be correlated. We highlight that the spatial channel decorrelation and the channel reciprocity assumptions are crucial for the security of WiKE.

**Remark 5.** Our consideration of both PPT and unbounded adversaries results in two definitions for WiKE of different strengths. To achieve unbounded WiKE security, it becomes clear that one must use unconditionally-secure codes to authenticate messages [36] instead of a computationally-secure MAC.

**Remark 6.** By including `Corrupt` query and defining condition (3) within our freshness definition we capture the forward secrecy property. This property guarantees the long-term secrecy of the session keys even in the event of a later com-

---

<sup>4</sup> In practical terms, this distance must be at least 6.25 cm for a wireless transmission occurring at 2.4 GHz.

promise of the pre-established authenticated channel. Intuitively, most WiKE protocols should satisfy forward secrecy since long-term secrets in WiKEs are used solely for message authentication during WiKE execution and not message confidentiality. Note that we could make our definition tighter by making unrefresh party instances (and corresponding partners) that are directly targeted with **Corrupt** query. Instead, we opted for a simple but more encompassing definition inspired by [6]. Namely, our definition is agnostic to the long-term setup type while capturing a meaningful security property.

## 4 WiKE-then-PAKE Security Model and Composition

Previously, we defined a game-based security model for Wireless-channel Key Exchange (WiKE) which considers WiKE in isolation. In this section, we aim to solve the problem of the slow rate of secret key generation that may occur because of inadequate environmental conditions. The main idea is to bootstrap a low-entropy secret coming from WiKE using Password Authenticated Key Exchange (PAKE). We propose a generic solution building on top of our WiKE security model: we define a compositional WiKE-then-PAKE model by following the techniques from [12] and [35]. Then, we prove that the composition of any WiKE protocol that is secure according to our WiKE model and any PAKE protocol that is secure in the standard Real-or-Random PAKE model is secure under our WiKE-then-PAKE model of security.

### 4.1 The Slow-rate Key Generation Problem

The goal of WiKE is to generate a secret key stream of high entropy and uniform distribution in the presence of an unbounded adversary. One important metric when assessing the utility of WiKE protocol is the secret key rate. This metric tells us how many secret bits/second (bps) we can expect to derive from WiKE protocol execution. This rate depends on many parameters such as the proposed WiKE method, indoor or outdoor environment, endpoint (node) mobility, the distance between sender and receiver nodes, the presence of different interference sources, etc. From various experimental results [23,39,33], we see that for particular WiKE protocols secret key rate range from 0.5 bps in static environments up to 15 bps in a highly dynamic outdoor setting. This means that in real-world conditions it may take from 15 seconds to a whole 8 minutes to generate a 256-bit secret key. We would argue that for some applications this observed latency is too high. Therefore, we pose the following question: How to quickly establish a secure session key in case of a slow key generation in WiKE protocols?

### 4.2 Solution

WiKE offers strong security guarantees – in our security model, we consider a powerful adversary with unbounded computational power and in physical proximity of either honest party. In normal environmental conditions, one can directly

use WiKE to obtain a session key that can be used for various applications (e.g., to establish a secret channel). However, depending on multiple factors linked to the environment, the WiKE protocol might be slow. One possible solution to deal with this slow key bitrate is the following: First, use WiKE to generate a secret bitstream during a pre-specified time period depending on the application, and then, as a fail-over mechanism, use a password-authenticated key exchange (PAKE) in case of a low-entropy output from WiKE to derive a high-entropy session key. In the rest of this section, we explore how to combine WiKE and PAKE and what security guarantees one might expect of such a composition.

**Design choices.** We consider two different realizations of a sequential WiKE-then-PAKE composition: (a) To establish a session key, WiKE protocol is followed by PAKE protocol. The high-entropy key output by PAKE can be used to secure a single session, or it can be stored to be used across multiple sessions. To refresh the key, the two parties engage in a new WiKE-then-PAKE protocol. (b) The two parties run once the WiKE protocol and store the output of WiKE as a long-term secret both parties share. Every time the two parties wish to establish a secure channel, they run PAKE to obtain a session key.

Arguments can be made to support one design choice over the other. If we were to store one key, we would opt for the high entropy that comes out of PAKE for the simple reason that in practical terms it would give fewer opportunities to an adversary to monitor, intercept and replace messages to attack the PAKE protocol. This choice is reflected in our compositional security model, as it means that the key that comes out of WiKE is just an ephemeral state of the instance, and therefore is not considered corruptible information. Looking forward, our choice reduces PAKE security requirements within the composition. Namely, instead of relying on Real-or-Random PAKE model with *perfect forward secrecy* (pfs-RoR) (see [1]), we can resort to the weaker one-time-password-authenticated key exchange [32] or the original RoR model [3] without forward secrecy. The reason for this relaxed requirement is that passwords input to PAKE are not repeated across instances. And although the one-time-PAKE model is strictly enough for this composition – as low-entropy secrets coming out of WiKE are uniformly and independently sampled and to be used only once – we opted for the original RoR model without forward secrecy. The motivation for such a choice is two-fold: 1) most real-world PAKEs are analyzed within the original RoR model; 2) although one-time PAKE is enough, it does not bring efficiency benefits for a concrete instantiation when compared to a full-fledged PAKE protocol. We highlight that our original RoR model has only been slightly enhanced with `Reveal` query for simplicity of proof exposition. The two models (without forward secrecy) are equivalent up to a factor 2, as `Reveal` queries can be simulated via `Test` queries. This is the only change to the original model.

**Security guarantees.** Since the security of all PAKE protocols relies on various computational hardness assumptions (e.g., discrete log-based, RSA-based, lattice-based, etc.), guarantees offered by our WiKE-then-PAKE composition

will also be computational. In our composed protocol, WiKE is used for initial secret generation. The high-level protocol running WiKE will decide, based on WiKE’s output length, whether PAKE execution is needed. The security level achieved by PAKE will be determined by the security parameter  $\lambda$ .

### 4.3 Composed Protocol WiKE-then-PAKE

Previously, we defined WiKE protocol as a pair of algorithms  $(WGen, W)$ , and PAKE protocol (see Appendix A.1) as a pair of algorithms  $(PGen, P)$ . In a similar fashion, we now define our *composed protocol* as a pair of algorithms  $(CGen, C)$ .

We instantiate algorithm  $CGen$  as a  $WGen$ <sup>5</sup> and algorithm  $C$  as expected: First,  $C$  runs the WiKE protocol  $W$ . Whenever an instance only manages to obtain a low entropy session key after successfully running  $W$  (due to inadequate environmental conditions and/or insufficient time to generate a high entropy key), that key is passed as input to the PAKE protocol  $P$  afterwards. The task of algorithm  $C$  is to track the status of an instance through status flags and switch to the appropriate sub-algorithm when necessary. Note that WiKE protocol outputs are independent and uniformly distributed (and potentially of low entropy), which perfectly fits our assumption that passwords are uniformly sampled from  $\mathbb{D}_{pw}$  in the RoR PAKE model from Appendix A.2. The secrets generated by the  $WGen$  algorithm can be seen as the long-term keys to the composed protocol.

### 4.4 Security Model for WiKE-then-PAKE

Here we define a security model for the sequential composition of WiKE and PAKE protocol. With  $G^{com}$ , we will denote a security game for our composed protocol. An adversary  $\mathcal{A}$  interacts with a challenger  $C^{com}$  that keeps the appropriate secret information away from  $\mathcal{A}$  while administrating the security experiment of game  $G^{com}$ .

We will define our model using the techniques from [12] and [35]. The goal of the adversary is to distinguish real session keys from random keys in the composed protocol WiKE-then-PAKE. Naturally, the composed protocol will be broken if: (1) An adversary manages to obtain partial or complete information about a WiKE protocol output, or (2) An adversary makes a correct guess on WiKE output (with or without relying on information leakage from WiKE execution). Intuitively, it is clear that we cannot hope for the composed protocol to achieve a better security guarantee than one coming from a PAKE protocol itself.

**Participants.** Without loss of generality, we will assume that the composition of WiKE-then-PAKE algorithms uses WiKE’s participant format of nodes. As a result,  $\mathbb{I}_{wike}$  and  $\mathbb{I}_{com}$  are equal. Interestingly, due to the particular way of defining password setup in RoR-PAKE model, where each client may only hold a single password, we will need to initiate a new PAKE client party for every

<sup>5</sup> Note that  $CGen$  also includes part of  $(PGen$  that is responsible for public parameter generation, but without password generation algorithm.

initiator instance of WiKE<sup>6</sup>. This issue does not occur on the responder side, as a server in RoR PAKE may hold many passwords for different clients.

**Protocol execution.** The protocol C is a PPT algorithm describing the reaction of principals to incoming messages from both physical and upper network layers. The adversary  $\mathcal{A}$  has the freedom to interact with multiple different executions of composed protocol C. We denote by  $U^i$  the  $i$ -th instance of principal  $U$  running C. In places where it matters, we will denote initiator instances  $T^i$  and responder instances  $R^j$ .

**Execution state of a principal instance.** The challenger  $\mathcal{C}^{com}$  will maintain the execution and game state for  $G^{com}$  and run initialization procedures similar to those in models for WiKE and PAKE. The execution state of the composed protocol contains all the necessary data for the actual executions of a WiKE protocol W in the first stage and a PAKE protocol P in the second stage.

Similarly to our WiKE model, execution state of each instance of our composed protocol C can be described as a tuple  $(U.setup, U^i.pid, U^i.sid, U^i.key, U^i.status, U^i.internal)$ , where all the execution state variables keep the same purpose. In the composed model, we use  $U.setup$  to store the long-term secrets from WiKE, and  $U^i.internal$  to store the low-entropy output of WiKE, which is an intermediary, ephemeral value used as a password input for PAKE.  $U^i.key$  now corresponds to the session key coming out of PAKE. The set of possible values for  $U^i.status$  now applies to the session key corresponding to the PAKE stage of execution. The session identifiers in the composed protocol will – in addition to the full transcript of W.Phase2 and W.Phase3 of WiKE – also include the full PAKE transcript. Various session and partner identifiers and other flags that track execution and game state will be handled appropriately.

**The network adversary.** Similar to WiKE and PAKE models, an adversary  $\mathcal{A}$  against game  $G^{com}$  has access to a set of queries via a standard game interface provided by the challenger. Queries from this set will correspond to a query or a combination of queries from both  $G^{wi\!ke}$  and  $G^{pa\!ke}$ . Thus, while playing  $G^{com}$ ,  $\mathcal{A}$  has a following set of queries:

**Execute( $T^i, R^j$ )** This query models a honest run of C between initiator  $T^i$  and responder  $R^j$ . The complete transcript of upper-layer communication (i.e. information reconciliation and privacy amplification phases from WiKE and the whole transcript from PAKE) is given to  $\mathcal{A}$ . As a result, instances compute the same high-entropy  $T^i.key = R^j.key \in \{0, 1\}^\lambda$  and status is updated  $T^i.status = R^j.status = terminated$ .

**Probe( $T^i, R^j$ )** This query is handled in the same way as in our WiKE model. It models an honest run of W.Phase1 (advantage creation phase) between initiator  $T^i$  and responder  $R^j$  of WiKE.

---

<sup>6</sup> This is a small manageable inconvenience that would not exist if one-time PAKE primitive is used.



- Send( $U^i, M$ )** This query models an active adversary for the phases W.Phase2 and W.Phase3 from WiKE and full PAKE protocol. As a result, a message  $M$  is sent to a principal instance  $U^i$  that responds to  $\mathcal{A}$  according to the protocol. Note that  $\mathcal{A}$  will be notified in case of successful WiKE completion, as well as in case instance  $U^i$  accepts or terminates its execution.
- Reveal( $U^i$ )** As a response to this query,  $\mathcal{A}$  receives the current value of the session key  $U^i.\text{key}$ .  $\mathcal{A}$  may ask this query only if  $U^i$  is successfully terminated (holding a session key) and a **Test** query has not been made to  $U^i$  or its partner instance.
- Corrupt( $U$ )** This query reveals secret setup of WiKE (and not the ephemeral low-entropy value ( $U^i.\text{internal}$ ) used as input to PAKE as discussed in 4.2).
- Test( $U^i$ )** At the beginning of  $G^{\text{com}}$ , a hidden bit  $b$  is randomly selected by  $\mathcal{C}^{\text{com}}$  and used to answer *all* **Test** queries. If  $b = 0$ ,  $U^i.\text{key}$  is given, otherwise a random key is sampled. As in WiKE and PAKE, consistency of answers is managed by  $\mathcal{C}^{\text{com}}$ .

**Partnering.** This definition is the same as the corresponding definition from our WiKE model (see Section 3.3).

**Freshness.** An instance  $T^i$  and its partner instance  $R^j$  are made *fresh* after **Execute( $T^i, R^j$ )** query is asked. Furthermore, an instance  $U^i$  (whether this is  $T^i$  or  $R^j$ ) that has accepted as a result of appropriate **Probe** and **Send** queries is *fresh* unless any of the following conditions hold: (1) **Reveal( $U^i$ )** query was asked previously, or (2) if **Reveal( $V^j$ )** query was asked previously where  $V^j$  is  $U^i$ 's partner instance, or (3) if any participant  $Q$  was target of **Corrupt( $Q$ )** query before  $U^i$  defined its ephemeral WiKE key stored in  $U^i.\text{internal}$ , and a **Send( $U^i, M$ )** query occurred.

**Security of the sequential composition.** As we asserted above, the security game of our composition  $G^{\text{com}}$  is inherently linked to the security game of PAKE  $G^{\text{pake}}$ . Formally, the advantage of  $\mathcal{A}$  in breaking the *com*-security between WiKE and PAKE is defined as

$$Adv_{\mathcal{C}}^{\text{com}}(\mathcal{A}) \stackrel{\text{def}}{=} |2 \cdot \Pr[b = b'] - 1|, \quad (3)$$

where  $b$  is the hidden bit selected at the beginning of  $G^{\text{com}}$ ,  $b'$  is adversary's choice, while  $\mathbb{P}[b' = b]$  is the probability of  $\mathcal{A}$  guessing the hidden bit  $b$ . As we saw before, it is clear that the composed protocol will inherit the limitations of underlying WiKE and PAKE protocols. Its security will, to the greatest extent, depend on the quality of the session key generated by WiKE, which is parameterized by  $\kappa$ . Further, WiKE produces keys that are information-theoretically indistinguishable from truly random keys, even considering an active adversary in phases 2 and 3. This maps particularly well to the assumption of RoR-security for PAKE that passwords are selected uniformly at random from a dictionary. Therefore, the "quality" of the key will only impact the dictionary size.

Therefore, the best that we can expect is to declare *com*-secure if there exists some positive constant  $B$  such that the *com*-advantage of  $\mathcal{A}$  in breaking  $\mathsf{C}$  satisfies

$$Adv_{\mathsf{C}}^{com}(\mathcal{A}) \leq \frac{B \cdot n_{se}}{|\mathbb{D}_{\kappa}|} + \epsilon(\kappa) + \epsilon(\lambda), \quad (4)$$

where  $n_{se}$  is an upper bound on the number of **Send** queries  $\mathcal{A}$  makes in  $G^{com}$ , function  $\epsilon$  is negligible function in its input length. Note that ideally  $B = 1$ , meaning at most one password guess per **Send** query.

#### 4.5 Black-box Composition Result

Here we present our composition results. We show in Theorem 1 that RoR-secure wireless-channel key exchange protocol securely composes with RoR-secure password-authenticated key exchange (without forward secrecy).

**Theorem 1.** *Let  $(\mathsf{WGen}, \mathsf{W})$  be a wireless-channel key exchange secure protocol according to Definition 2 that outputs keys in key space  $\mathbb{D}_{\kappa}$ . Let  $(\mathsf{PGen}, \mathsf{P})$  be a password-authenticated key exchange protocol secure according to Definition 11. The composed protocol  $(\mathsf{CGen}, \mathsf{C})$  such that  $\mathsf{CGen} \stackrel{\text{def}}{=} \mathsf{WGen}$  and  $\mathsf{C} \stackrel{\text{def}}{=} \mathsf{P} \circ \mathsf{W}$  (as described in detail in Subsection 4.3) is secure according to the composition game  $G^{com}$ , and the advantage of any efficient adversary  $\mathcal{A}$  against the composed protocol  $(\mathsf{CGen}, \mathsf{C})$  satisfies the inequality*

$$Adv_{\mathsf{C}}^{com}(\mathcal{A}) \leq 2 \cdot Adv_{\mathsf{W}}^{wike}(\mathcal{B}_1) + Adv_{\mathsf{P}}^{pake}(\mathcal{B}_2) \quad (5)$$

for some PPT adversaries  $\mathcal{B}_1$  and  $\mathcal{B}_2$ . Furthermore, the advantage of  $\mathcal{B}_1$  is parameterized by a security parameter  $\kappa$ , the advantage of  $\mathcal{B}_2$  is parameterized by security parameter  $\lambda$  and WiKE output key space  $\mathbb{D}_{\kappa}$ .

Below we provide the proof sketch, while the detailed proof of Theorem 1 can be found in the full version of this paper.

*Proof (Theorem 1).* Let us fix a PPT adversary  $\mathcal{A}^{com}$  attacking the protocol  $\mathsf{C}$ . Let  $\mathsf{G}_x$  be the event that  $\mathcal{A}^{com}$  outputs 1 in **Game  $\mathsf{G}_x$** . We will exhibit our proof as a sequence of four games to bound the advantage of  $\mathcal{A}^{com}$  against  $\mathsf{C}$ .

**Game  $\mathsf{G}_0$  (The original game with  $b = 0$ , i.e. real keys)** Let this be the game as defined in Section 4.4 for the composed protocol  $\mathsf{C}$  that is built as described in Section 4.3 with a fixed challenge bit  $b = 0$ . Whenever  $\mathcal{A}^{com}$  queries  $\text{Test}^{com}(U^i)$  oracle, the real session key  $U^i.\text{key}$  is provided.

**Game  $\mathsf{G}_1$  (WiKE output random)** Whenever  $\mathcal{A}^{com}$  queries  $\text{Execute}^{com}$  or  $\text{Send}^{com}$  that successfully completes  $\mathsf{W.Phase3}$  of the WiKE part of the composed protocol, the ephemeral key coming out of the WiKE set in the internal state of the instance is replaced with a randomly sampled key of the same length, except in the two cases identified below. The protocol then continues the execution with

this key used as a password for PAKE, whether in the remaining steps necessary to conclude  $\text{Execute}^{com}$  or in the  $\text{Send}^{com}$  queries that follow.

Case 1 – In case there is another instance  $V^j$  whose ephemeral WiKE key is already set and has the same session identifier, i.e.  $U^i.\text{sid} = V^j.\text{sid}$ , then we set the ephemeral WiKE key in the internal state of  $U^i$  to match that of  $V^j$ . Note that the ephemeral key stored in  $V^j.\text{internal}$  is random anyway, and this case is just for consistency unless Case 2 happened.

Case 2 – If  $\mathcal{A}^{com}$  queries  $\text{Send}^{com}(U^i)$  that successfully completes  $\text{W.Phase3}$  and previously asked a  $\text{Corrupt}^{com}$  query, in which case the adversary might force an authenticated message to  $U^i$ , the ephemeral key from WiKE cannot be replaced.

The distance between  $\mathbf{G}_0$  and  $\mathbf{G}_1$  is bounded by the advantage against WiKE.  $\mathcal{C}^{com}$  uses an adversary  $\mathcal{B}_1$  against WiKE that helps  $\mathcal{C}^{com}$  interpolate between the two games.  $\mathcal{B}_1$  makes use of  $\text{Test}^{com}$  queries in the WiKE game to get WiKE ephemeral keys, either real ones matching the description of  $\mathbf{G}_0$  or random ones, matching the description of  $\mathbf{G}_1$ . To deal with Case 2,  $\mathcal{B}_1$  makes use of the  $\text{Reveal}^{wike}$  query provided by the WiKE game.  $\text{Corrupt}^{com}$  queries to  $\mathcal{C}^{com}$  are passed on to  $\mathcal{B}_1$  to get the answer. Notice that  $\mathcal{B}_1$  never asks unfresh  $\text{Test}^{wike}$  queries as these will fall precisely in Case 2.

$$| \Pr[\mathbf{G}_1] - \Pr[\mathbf{G}_0] | \leq \text{Adv}_{\mathbf{W}}^{wike}(\mathcal{B}_1) \quad (6)$$

**Game  $\mathbf{G}_2$  (PAKE output random)** In this game, whenever  $\mathcal{A}^{com}$  asks a  $\text{Test}^{com}(U^i)$  query, a random session key  $U^i.\text{key}$  is sampled, keeping track of partnerships for consistency.  $\mathcal{C}^{com}$  creates an algorithm  $\mathcal{B}_2$  that plays against PAKE and helps  $\mathcal{C}^{com}$  interpolate between  $\mathbf{G}_1$  and  $\mathbf{G}_2$ . Whenever  $\mathcal{A}^{com}$  asks a  $\text{Test}^{com}$  query, this is passed on to  $\mathcal{B}_2$  that places a  $\text{Test}^{pake}$  query against the PAKE game. All passwords are uniformly distributed, as per description of  $\mathbf{G}_1$ , except whenever a  $\text{Corrupt}^{com}$  query previously occurred. But in that case, all interactions with that party instance are computed by  $\mathcal{C}^{com}$  without relaying the messages to  $\mathcal{B}_2$ . In any case, if  $\text{Corrupt}^{com}$  occurred, parties are unfresh and the adversary cannot ask a  $\text{Test}^{com}$  query. The distance between  $\mathbf{G}_1$  and  $\mathbf{G}_2$  is bounded by the advantage of  $\mathcal{B}_2$  against PAKE.

$$| \Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_1] | \leq \text{Adv}_{\mathbf{P}}^{pake}(\mathcal{B}_2) \quad (7)$$

**Game  $\mathbf{G}_3$  (WiKE output real, PAKE output random, the original game with  $b = 1$ )** In this game, we revert the change made in  $\mathbf{G}_1$  and whenever  $\mathcal{A}^{com}$  queries  $\text{Execute}^{com}$  or  $\text{Send}^{com}$  that successfully completes  $\text{W.Phase3}$  of the WiKE part of the composed protocol, the actual ephemeral key coming out of WiKE is used in the rest of the protocol. Again, the distance between  $\mathbf{G}_2$  and  $\mathbf{G}_3$  is bounded by the advantage of  $\mathcal{B}_1$  against the WiKE game.

$$| \Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_2] | \leq \text{Adv}_{\mathbf{W}}^{wike}(\mathcal{B}_1) \quad (8)$$

Notice that  $\mathbf{G}_3$  is as described in Section 4.4 with bit  $b = 1$ , i.e. whenever  $\mathcal{A}^{com}$  asks  $\text{Test}^{com}(U^i)$  the real session key  $U^i.\text{key}$  is provided. By combining Eq. 6, 7, and 8 we obtain the equations 9. This concludes the proof of Theorem 1.

$$Adv_{\mathcal{C}}^{com}(\mathcal{A}) \stackrel{\text{def}}{=} |\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_0]| \leq 2 \cdot Adv_{\mathbf{W}}^{wike}(\mathcal{B}_1) + Adv_{\mathbf{P}}^{pake}(\mathcal{B}_2) \quad (9)$$

**Secure instantiation of composition between WiKE and PAKE.** As a direct consequence of Theorem 1, one can securely instantiate our composed protocol from Section 4.3 with any WiKE protocol that meets Definition 2 and any PAKE protocol that meets Definition 11, thereby obtaining the security guarantees from Theorem 1. We leave for future work the security analysis of concrete WiKE schemes within our model. Thus, we can not give definite advice on concrete WiKE instantiation. We refer the reader to six concrete WiKE protocols that have empirically been tested in comprehensive experiments in [33]. Regarding PAKE instantiation, we believe that there exist mature and robust balanced PAKE protocols such as SPAKE2 [5,1] or CPace[4] that can be used in a WiKE-then-PAKE configuration. For more information on state-of-the-art PAKE protocols, we refer the reader to [21].

## 5 On the Security of vPAKE Protocol

The authors from [41] propose a custom-tailored PAKE called vPAKE (see Figure 1) that aims at establishing a secret session key from a low-entropy secret coming from WiKE. As we showed in the previous section of this paper, a regular PAKE not only is sufficient for the job, it does not even need to be *forward secure*.

Here, we show that the security proof of vPAKE in the FtG model [6] provided in [41] is unconvincing since it falls into a circular argument. Of independent interest is an attack on the vPAKE protocol that allows an attacker to check if a target user registered the same password with two different servers. In all fairness to the authors, such an attack is benign if the actual password is *fresh* from WiKE, and it is not covered by the FtG model from [6] because within the model each client has a single password that is registered with every server. Interestingly, even in more recent adaptations of the RoR model where unique passwords are sampled per client-server pair [1], although such an attack is possible, the strategy of looking for repeated passwords yields no benefit to an adversary within the model when compared to the naive approach of trial guessing from the dictionary: both strategies costs at least one **Send** query per trial-guess/password-reuse-test. In the real world, password reuse is a real phenomenon and such a vulnerability has real implications. (It is noteworthy to mention that this attack is captured by stronger notions of PAKE defined within the UC framework.) In the rest of this section, we explain in detail why the security proof from [41] falls into a circular argument and how an attacker

Public parameters:	$g_1, g_2, h \in \mathbb{G};$	$H : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda;$
<b>User X</b> (input : pw)	<b>User Y</b> (input : pw)	
$x \leftarrow \mathbb{Z}_q$	$y \leftarrow \mathbb{Z}_q$	
$A := g_1^x h^{\text{pw}}$	$B := g_1^y h^{\text{pw}}$	
	$\xrightarrow{A}$	$\xleftarrow{B}$
$C := B/h^{\text{pw}}$	$E := A/h^{\text{pw}}$	
$D := C^x$	$F := E^y$	
$L := H(\text{id}_X    (A \oplus B)    D)$	$J := H(\text{id}_Y    (B \oplus A)    F)$	
$U := x \oplus L$	$V := y \oplus J$	
	$\xrightarrow{U}$	$\xleftarrow{V}$
$J' := H(\text{id}_Y    (A \oplus B)    D)$	$L' := H(\text{id}_X    (B \oplus A)    F)$	
$M := V \oplus J'$	$N := U \oplus L'$	
if $g_1^M == C$	if $g_1^N == E$	
$\text{sk} := (g_2^M)^x$	$\text{sk} := (g_2^N)^y$	
return sk	return sk	
else return $\perp$	else return $\perp$	

**Fig. 1.** Protocol vPAKE [41]

would test for password reuse<sup>7</sup>.

**Obstacles in proving vPAKE secure: A circular argument.** A careful analysis of the security proof provided by [41] reveals that a game hop crucial for proving the security of the protocol cannot be reduced to the DDH problem as claimed because the argument falls into a fallacy of circular reasoning. Here we dive into the details of the proof and pinpoint exactly where and why the argument does not hold.

- Game  $G_0$ : This is the original Find-then-Guess game [6] for the proposed vPAKE protocol.

<sup>7</sup> Note that in the FtG model [6], should a **Send** query result in a party instance accepting, this event is made visible to the adversary. However, in the original protocol from Zhang et al. [41], in the key confirmation round, instead of rejecting unsuccessful session, the protocol samples new non-matching random keys and continues. It's unclear when the protocol accepts and why would a party terminate with a non-matching key, which is bound to fail when used in any meaningful way. Therefore, we modify the protocol to reject when the key confirmation round fails.

- Game  $G_1$ : When the adversary queries the `Execute` oracle, the challenger samples the password from  $\mathbb{Z}_q \setminus \mathbb{D}$ , instead of sampling from dictionary  $\mathbb{D}$ . The distance between  $G_0$  and  $G_1$  cannot be zero as claimed by the authors as an hypothetical adversary with access to a CDH oracle can trivially distinguish between both games. However, the crucial problem with the proof comes in the next game hop.
- Game  $G_2$ : When the adversary queries the `Execute` oracle, the challenger samples  $U$  and  $V$  uniformly at random. (The authors sample  $s', t' \leftarrow \mathbb{Z}_q$  and set  $U := s' \oplus L$  and  $V := t' \oplus J$ , but ultimately claim that  $U$  and  $V$  look random.)

Analysis of game hop  $G_1 \rightarrow G_2$ : We only look at the left side of the protocol for simplicity. In order for  $U := x \oplus L$  to look random, it must be that either  $x$  or  $L$  (or both) are random and independent from all other values. Exponent  $x$  is definitely random but not independent as it is used throughout the protocol.  $L$  is random if  $H(id_x, A \oplus B, D)$  is modeled as a random oracle. But is it independent as well?  $H$  being a random oracle,  $L$  is random and independent if the adversary never queries  $H$  on the exact same input  $(id_x, A \oplus B, D)$ . One would be tempted to argue that it is hard for an adversary to compute  $D := g^{xy}$  as it seems to boil down to solving the CDH.

The authors of [41] reduce the distance between  $G_1$  and  $G_2$  to the DDH assumption, arguing that  $D$  looks random from the adversary point-of-view, and therefore the likelihood of adversary  $\mathcal{A}$  querying  $H(id_x, A \oplus B, D)$  is small. To formalize this intuition, one has to show that an adversary  $\mathcal{A}$  that wins with noticeably more probability in  $G_2$  compared to  $G_1$  can be used by an adversary  $\mathcal{B}$  to distinguish a DDH tuple. Adversary  $\mathcal{B}$  receives  $(X := g^x, Y := g^y, D)$  and is asked to decide whether  $D = g^{xy}$  or  $D \leftarrow \mathbb{Z}_q$ . To do so, it embeds the challenge  $(X, Y, D)$  received from its own game wherever it needs to compute  $g^x$ ,  $g^y$  and  $D$ , and tries to simulate the game  $\mathcal{A}$  is playing.

All looks good until  $\mathcal{B}$  has to complete the simulation of the `Execute` query for  $\mathcal{A}$  and compute  $U := x \oplus L$ . Notice that  $\mathcal{B}$  received  $X := g^x$ , cannot compute  $x$ , and  $x$  is still necessary to simulate the completion of `Execute` query. Granted that the whole point is to make  $U$  random, in which case  $x$  is not needed, but then the argument becomes circular.

It doesn't mean there's an obvious attack to the protocol, but the reduction is flawed and one cannot claim provable security either. This is similar to encrypting the decryption key under the public key. Most public-encryption schemes are not obviously broken if one encrypts the decryption key under the corresponding public-key, but designing provably secure encryption schemes in this settings is known to be challenging [10]. Splitting this game hop into smaller steps would look like this:

- $G_{2a}$ : In this game, the `Execute` query sets  $D \leftarrow \mathbb{Z}_q$ . If the remaining operations of the protocol didn't require the explicit knowledge of  $x$ , we could embed the DDH challenge in the `Execute` query, and the distance from this

game to  $G_1$  would be bound by the advantage of distinguishing a DDH tuple. Unfortunately, it is not the case in this variant of the protocol and this is precisely where the argument breaks.

- $G_{2b}$ : In this game, we set a bad event whenever  $\mathcal{A}$  queries  $H(id_x, A \oplus B, D)$ . Since  $D$  is random and independent from everything else, the probability of this happening is statistically bound to  $\frac{n}{q}$ , where  $n$  is the number of queries  $\mathcal{A}$  places to the random oracle  $H(\cdot)$  and  $q$  is the order of  $\mathbb{Z}_q$ .
- $G_{2c}$ : In this game, the Execute query sets  $L \leftarrow \mathbb{Z}_q$ . Because  $H(\cdot)$  is modeled as a random oracle and  $\mathcal{A}$  never queries  $H(id_x, A \oplus B, D)$ , nothing changes.
- $G_{2d}$ : In this game, the Execute query sets  $U \leftarrow \mathbb{Z}_q$ . This is transparent for  $\mathcal{A}$  because  $L$  is random and independent.

The above broken argument does require  $H(\cdot)$  to be modeled as a random oracle (collision resistance is not sufficient to secure the one-time pad), but the random oracle does not have to be programmable. Alternative reduction to Gap-CDH (as in [1] to prove the security of SPAKE2, the PAKE this protocol is based on), or even CDH (with loss of tightness) could be considered but require  $H(\cdot)$  to be modeled as a random oracle with programmability. However, we restrict our attention to eavesdropping adversaries only as this is enough to show that the claim does not hold.

**Password reuse attack.** Standard game-based definitions for PAKE protocols, such as those known as Find-then-Guess (FtG) [6] and Real-or-Random (RoR) [3], are known not to capture adversarial attacks that exploit relations between passwords. In real world scenarios, it is common for users to choose closely-related passwords, mistype passwords, or even reuse passwords in different services. On the other hand, security definitions in the Universal Composability framework [14] cover these attack vectors as well, reason why they have become the gold-standard for proving security of PAKE protocols [4,24].

Although vPAKE was designed to be used as an extension to the physical layer security [20], in which case it might be reasonable to assume that no such relations between passwords exist, it is worth noting that vPAKE is vulnerable to such attacks. In particular, we show how an attacker with intercept, redirect and replace capabilities over a network, can test if a user  $X$  registered the same password with server  $S_1$  and server  $S_2$ .

1. User  $X$  wants to authenticate with server  $S_1$ .  $X$  sends  $(X, S_1, A)$  as a message from  $X$  to  $S_1$ .
2. Adversary  $\mathcal{A}$  intercepts the message, and forwards  $(X, S_2, A)$  to server  $S_2$ .
3. Server  $S_2$  thinks user  $X$  wants to authenticate and replies  $(S_2, X, B)$  to  $X$ .
4. Adversary  $\mathcal{A}$  intercepts the message, and sends  $(S_1, X, B)$  to user  $X$ .
5. User  $X$  thinks he received a reply from  $S_1$  since he initiated the protocol and replies  $(X, S_1, U)$  to server  $S_1$ .
6. Adversary  $\mathcal{A}$  intercepts the message and forwards  $(X, S_2, U)$  to  $S_2$ .

7. If server  $S_2$  accepts, the password that  $X$  used for authentication with server  $S_1$  is the same password registered with  $S_2$ .

This attack was possible with vPAKE because the protocol does not strictly bind both sender and receiver identities in hash function  $H(\cdot)$ , as in the original SPAKE2 protocol [5] it is based on. Another related problem is that server  $S_2$  can be left hanging, expecting further engagement from user  $X$ , and possibly resulting in a denial of service attack.

## 6 Conclusion and Future Directions

We proposed a security model for WiKE in the style of [3], which provides clarity on the security guarantees of WiKE, and allows us to compose WiKE with other cryptographic primitives within a formal provable security framework. By doing so, we showed how PAKE can be used to solve the problem of slow key rate in WiKE. As a result of successfully completing the third phase, the parties are able to agree on a common secret even in the presence of an unbounded adversary, as long as it does not actively interfere during the probing phase or sit near either legitimate party.

In the Real-or-Random security model of PAKE, passwords are sampled uniformly, at random from the dictionary. The fact that passwords are usually selected by humans, and therefore rarely uniformly distributed, is often stressed as a weakness of the Real-or-Random model. The WiKE-then-PAKE construction does not have this problem since the PAKE input password is the WiKE output.

This work formally combines a three-phase WiKE with other cryptographic primitives, of which PAKE is the natural candidate. Other works focus on providing a better solution to privacy amplification and even information reconciliation phases via information-theoretic authenticated key exchange (IT-AKA) and robust fuzzy extractors [17]. These solutions admit active adversaries with unbounded computational power and do not assume an authenticated channel. The caveat is that secrets must be high-entropy enough to render offline dictionary attacks infeasible, which is precisely the problem we tackle here. An interesting open question is whether it is possible to run a two-phase WiKE (i.e. without the privacy amplification phase) and combine it with a UC-secure PAKE [15], or even a single-phase WiKE with a Fuzzy PAKE [19].

## Acknowledgements

We thank the anonymous reviewers of CT-RSA 2023 for their careful reading of our manuscript and their many insightful comments and suggestions. Afonso Ariaga and Marjan Škrobot were supported by the Luxembourg National Research Fund (FNR), under the CORE Junior project (C21/IS/16236053/FuturePass).



## A Security Model for PAKE

Today, the Real-or-Random (RoR) model from [3] and the Universally Composable PAKE model from [14] are considered state-of-the-art models rigorously capturing PAKE security requirements. In this paper, we will use a variant of the RoR definition from [3], where `Reveal` is added. `Reveal` query was available in the original Find-then-Guess model and removed later from the RoR because it can be simulated via `Test` oracle, which in the RoR model can be queried multiple times. However, having a `Reveal` oracle facilitates proof reductions that rely on the security of PAKE and was later adopted by multiple authors [2,35].

### A.1 PAKE Protocol

We represent PAKE protocol as a pair of algorithms (PGen, P). PGen is a password generation algorithm, while P defines the execution of the PAKE protocol. PGen samples passwords uniformly at random from the dictionary  $\mathbb{D}_{pw}$ . We assume that P describes several sub-algorithms, one of which is responsible for the generation of public parameters, common to all principals.

### A.2 Real-or-Random Security Model for PAKE

Let us denote a game that represents the RoR security model  $G^{pake}$ . For such a game, there exists a challenger  $\mathcal{C}^{pake}$  that will keep the appropriate secret information away from an adversary  $\mathcal{A}$  while administrating the security experiment. We denote the security parameter by  $\lambda \in \mathbb{N}$ .

**Participants and passwords.** For the two-party PAKE scenario, each principal  $U$ , identified by a string, comes either from a client set  $\mathbb{C}$  or a server set  $\mathbb{S}$ , which are finite, disjoint, nonempty sets. We denote the union of  $\mathbb{C}$  and  $\mathbb{S}$  sets as  $\mathbb{I}_{pake}$ . As usual, we assume that each client  $C \in \mathbb{C}$  possesses a password  $C.pw$ , while each server  $S \in \mathbb{S}$  holds a vector of the passwords of all clients  $S.PW := \langle C.pw \rangle_{C \in \mathbb{C}}$ . We assume that these passwords are sampled independently and uniformly from  $\mathbb{D}_{pw}$  at the start of  $G^{ror}$ .

**Protocol execution.** The protocol P is a PPT algorithm that describes the reaction of principals to incoming messages. In our model, we allow each principal to run an unlimited number of *instances* to model real-world parallel executions of P. We denote  $U^i$  the  $i$ -th instance of principal  $U$ . In places that matters, we will denote initiator instances  $C^i$  and responder instances  $S^j$ .

**Full network adversary.** When analyzing the security of P, we assume that our adversary  $\mathcal{A}$  has complete network control.  $\mathcal{A}$  has access to principals' instances via `Execute`( $C^i, S^j$ ), `Send`( $U^i, M$ ), `Reveal`( $U^i$ ), and `Test`( $U^i$ ) queries provided by  $\mathcal{C}^{pake}$ . These are standard RoR PAKE model queries as described in [6,3] that  $\mathcal{A}$  may ask multiple times (even `Test` queries).

**Initialization and internal state.** The challenger  $C^{pake}$  maintains *execution state* and *game state* in order to run a sound simulation. In an initialization phase, public parameters and the internal state are fixed. The appropriate sub-algorithm of  $P$ , called  $PGen$ , is run to generate the system’s public parameters. From the adversary’s perspective, an instance  $C^i$  comes into being after  $Send(C^i, S)$  query is asked. For each client a secret  $C.pw$  is drawn uniformly and independently at random from a finite set  $\mathbb{D}_{pw}$  of size  $|\mathbb{D}_{pw}|$ .

**Partnering.** We say that instance  $C^i$  is a partner instance to  $S^j$  and vice versa if: (1)  $C$  is a client and  $S$  is a server or vice versa, (2)  $sid := C^i.sid = S^j.sid \neq \perp$ , (3)  $C^i.pid = S$  and  $S^j.pid = C$ , (4)  $C^i.key = S^j.key$ , and (5) no other instance has a non- $\perp$  session identity equal to  $sid$ .

**Freshness.** An instance becomes *fresh* once it accepts (with or without a partner). An instance  $U^i$  then becomes *unfresh* if any of the following events occurs: (1)  $Reveal(U^i)$  query is asked, (2) if  $Reveal(V^j)$  query is asked and  $V^j$  is  $U^i$ ’s partner instance.

**PAKE security.** Now we can formally define RoR PAKE advantage of  $\mathcal{A}$  against  $P$ . At some point in time,  $\mathcal{A}$  will end  $G^{pake}$  and outputs a bit  $b'$ . We say that  $\mathcal{A}$  wins and breaks the RoR security of  $P$  if  $b' = b$  ( $b$  being the hidden bit selected at the beginning of  $G^{pake}$ ). The probability of this event is denoted by  $\Pr[b' = b]$ . The *pake*-advantage of  $\mathcal{A}$  in breaking  $P$  is defined as

$$Adv_P^{pake}(\mathcal{A}) \stackrel{\text{def}}{=} |2 \cdot \Pr[b = b'] - 1|. \quad (10)$$

Finally, we say that  $P$  is *pake*-secure if there exists a positive constant  $B$  such that for every PPT adversary  $\mathcal{A}$  it holds that

$$Adv_P^{pake}(\mathcal{A}) \leq \frac{B \cdot n_{se}}{|\mathbb{D}_{pw}|} + \epsilon(\lambda), \quad (11)$$

where  $n_{se}$  is an upper bound on the number of  $Send$  queries  $\mathcal{A}$  makes,  $|\mathbb{D}_{pw}|$  is the cardinality of  $\mathbb{D}_{pw}$ , and function  $\epsilon$  is negligible in the security parameter  $\lambda$ . Moreover, passwords are assigned uniformly at random to clients.

## References

1. Abdalla, M., Barbosa, M., Bradley, T., Jarecki, S., Katz, J., Xu, J.: Universally Composable Relaxed Password Authenticated Key Exchange. In: CRYPTO 2020. LNCS, vol. 12170, pp. 278–307. Springer (2020)
2. Abdalla, M., Benhamouda, F., MacKenzie, P.: Security of the J-PAKE Password Authenticated Key Exchange Protocol. In: S&P 2015. pp. 571–587. IEEE Computer Society (2015)
3. Abdalla, M., Fouque, P., Pointcheval, D.: Password-Based Authenticated Key Exchange in the Three-Party Setting. In: PKC 2005. LNCS, vol. 3386, pp. 65–84. Springer (2005)

4. Abdalla, M., Haase, B., Hesse, J.: Security Analysis of CPace. In: ASIACRYPT 2021. LNCS, vol. 13093, pp. 711–741. Springer (2021)
5. Abdalla, M., Pointcheval, D.: Simple Password-Based Encrypted Key Exchange Protocols. In: CT-RSA 2005. LNCS, vol. 3376, pp. 191–208. Springer (2005)
6. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated Key Exchange Secure Against Dictionary Attacks. In: EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer (2000)
7. Bellare, M., Rogaway, P.: Entity Authentication and Key Distribution. In: Advances in Cryptology — CRYPTO 1993. pp. 232–249. Springer (1994)
8. Bellare, M., Merritt, M.: Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In: S&P 1992. pp. 72–84. IEEE Computer Society (1992)
9. Bindel, N., Brendel, J., Fischlin, M., Goncalves, B., Stebila, D.: Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange. In: PQCrypto 2019. LNCS, vol. 11505, pp. 206–226. Springer (2019)
10. Boneh, D., Halevi, S., Hamburg, M., Ostrovsky, R.: Circular-Secure Encryption from Decision Diffie-Hellman. In: CRYPTO 2008. pp. 108–125. Springer (2008)
11. Boyko, V., MacKenzie, P.D., Patel, S.: Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. In: EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer (2000)
12. Brzuska, C., Fischlin, M., Warinschi, B., Williams, S.C.: Composability of Bellare-Rogaway Key Exchange Protocols. In: CCS 2011. pp. 51–62. ACM (2011)
13. Cachin, C., Maurer, U.M.: Linking Information Reconciliation and Privacy Amplification. *Journal of Cryptology* **10**(2), 97–110 (1997)
14. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.D.: Universally Composable Password-Based Key Exchange. In: EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer (2005)
15. Canetti, R., Krawczyk, H.: Universally composable notions of key exchange and secure channels. In: Knudsen, L.R. (ed.) *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings.* LNCS, vol. 2332, pp. 337–351. Springer (2002)
16. Csiszar, I., Korner, J.: Broadcast Channels with Confidential Messages. *IEEE Transactions on Information Theory* **24**(3), 339–348 (1978)
17. Dodis, Y., Wichs, D.: Non-Malleable Extractors and Symmetric Key Cryptography from Weak Secrets. In: STOC 2009. pp. 601–610. ACM (2009)
18. Dowling, B., Hansen, T.B., Paterson, K.G.: Many a Mickle Makes a Muckle: A Framework for Provably Quantum-Secure Hybrid Key Exchange. In: PQCrypto 2020. LNCS, vol. 12100, pp. 483–502. Springer (2020)
19. Dupont, P.A., Hesse, J., Pointcheval, D., Reyzin, L., Yakubov, S.: Fuzzy Password-Authenticated Key Exchange. In: EUROCRYPT 2018. pp. 393–424. Springer (2018)
20. Hamamreh, J.M., Furqan, H.M., Arslan, H.: Classifications and Applications of Physical Layer Security Techniques for Confidentiality: A Comprehensive Survey. *IEEE Communications Surveys Tutorials* **21**(2), 1773–1828 (2019)
21. Hao, F., van Oorschot, P.C.: SoK: Password-Authenticated Key Exchange - Theory, Practice, Standardization and Real-World Lessons. In: ASIA CCS 2022. pp. 697–711. ACM (2022)
22. Jakes, W.C.: *Microwave Mobile Communications.* Wiley/IEEE Press (1994)

23. Jana, S., Premnath, S.N., Clark, M., Kasera, S.K., Patwari, N., Krishnamurthy, S.V.: On the Effectiveness of Secret Key Extraction from Wireless Signal Strength in Real Environments. In: MOBICOM 2009. pp. 321–332. ACM (2009)
24. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-Computation Attacks. In: EUROCRYPT 2018. pp. 456–486. LNCS, Springer (2018)
25. Katz, J., Ostrovsky, R., Yung, M.: Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords. In: EUROCRYPT 2001. LNCS, vol. 2045, pp. 475–494. Springer (2001)
26. Mathur, S., Trappe, W., Mandayam, N.B., Ye, C., Reznik, A.: Radio-Telepathy: Extracting a Secret Key from an Unauthenticated Wireless Channel. In: MOBICOM 2008. pp. 128–139. ACM (2008)
27. Maurer, U.M., Wolf, S.: Information-Theoretic Key Agreement: From Weak to Strong Secrecy for Free. In: EUROCRYPT 2000. LNCS, vol. 1807, pp. 351–368. Springer (2000)
28. Maurer, U.M., Wolf, S.: Secret-Key Agreement Over Unauthenticated Public Channels – III: Privacy Amplification. *IEEE Transactions on Information Theory* **49**(4), 839–851 (2003)
29. Maurer, U.: Secret Key Agreement by Public Discussion from Common Information. *IEEE Transactions on Information Theory* **39**(3), 733–742 (1993)
30. Mosca, M., Stebila, D., Ustaoglu, B.: Quantum Key Distribution in the Classical Authenticated Key Exchange Framework. In: PQCrypto 2013. LNCS, vol. 7932, pp. 136–154. Springer (2013)
31. Mukherjee, A., Fakoorian, S.A.A., Huang, J., Swindlehurst, A.L.: Principles of Physical Layer Security in Multiuser Wireless Networks: A Survey. *IEEE Communications Surveys Tutorials* **16**(3), 1550–1573 (2014)
32. Paterson, K.G., Stebila, D.: One-Time-Password-Authenticated Key Exchange. In: ACISP 2010. LNCS, vol. 6168, pp. 264–281. Springer (2010)
33. Qu, Z., Zhao, S., Xu, J., Lu, Z., Liu, Y.: How To Test the Randomness From the Wireless Channel for Security? *IEEE Transactions on Information Forensics and Security* **16**, 3753–3766 (2021)
34. Renner, R., Wolf, S.: Simple and Tight Bounds for Information Reconciliation and Privacy Amplification. In: ASIACRYPT 2005. LNCS, vol. 3788, pp. 199–216. Springer (2005)
35. Skrobot, M., Lancrenon, J.: On Composability of Game-Based Password Authenticated Key Exchange. In: Euro S&P. pp. 443–457 (2018)
36. Wegman, M.N., Carter, L.: New Hash Functions and Their Use in Authentication and Set Equality. *Journal of Computer and Systems Sciences* **22**(3), 265–279 (1981)
37. Wyner, A.D.: The Wire-Tap Channel. *The Bell System Technical Journal* **54**(8), 1355–1387 (1975)
38. Xiao, L., Greenstein, L.J., Mandayam, N.B., Trappe, W.: Using the Physical Layer for Wireless Authentication in Time-Variant Channels. *IEEE Transactions on Wireless Communications* **7**(7), 2571–2579 (2008)
39. Ye, C., Mathur, S., Reznik, A., Shah, Y., Trappe, W., Mandayam, N.B.: Information-Theoretically Secret Key Generation for Fading Wireless Channels. *IEEE Transactions on Information Forensics and Security* **5**(2), 240–254 (2010)
40. Zhang, J., Duong, T.Q., Marshall, A., Woods, R.F.: Key Generation From Wireless Channels: A Review. *IEEE Access* **4**, 614–626 (2016)
41. Zhang, Y., Xiang, Y., Wu, W., Alelaiwi, A.: A Variant of Password Authenticated Key Exchange Protocol. *Future Generation Computer Systems* **78**, 699–711 (2018)