ZeroProKeS: A Secure Zeroconf Key Establishment Protocol for Large-Scale Low-Cost Applications

Shahnewaz Karim Sakib, George T Amariucai, Yong Guan

Abstract—Traditional approaches to authenticated key establishment include the use of PKI or trusted third parties. While certificate deployment is sub-optimal for large-scale, low-cost applications, the use of trusted third parties is subject to human error and leaked credentials. For this context, co-location can be a valuable resource, and it is often exploited through common randomness harvesting techniques, but these, in turn, suffer from low achievable rates and usually from restrictive assumptions about the environment. Recent techniques for exploiting co-location are based on the notion of quality time and rely on sophisticated throttled clue-issuing mechanisms that allow a device with enough time to spend in the vicinity of the transmitter to find a secret key by collecting enough consecutive clues. By contrast, attackers are afforded only limited time to listen to, or interact with, the clue transmitter. Previous work in this direction deals solely with passive attackers and uses high-overhead information throttling mechanisms. This paper introduces the active attacker model for the quality-time paradigm and proposes a simple solution, a Zeroconf Key Establishment Protocol (ZeroProKeS). Additionally, the paper shows how to efficiently expand the proposed protocol to adhere to any customized information transfer function between legitimate users.

Index Terms—Security Protocols, Zeroconf, Key Establishment, Set Intersection, Erasure Coding

1 Introduction

Establishing a secret key between two communicating parties lies at the root of any secure communication channel. While many such key establishment protocols exist, spanning a wide range of techniques and technologies, from public key infrastructure (PKI) to harvesting common randomness, very few appear to be appropriate for very-large-scale, low-cost deployment. Imagine the cost and time requirements for installing PKI certificates on hundreds or thousands of sensors that are awaiting together to be deployed en masse in the field in a total-loss manner (without the intention of recovering them once their batteries run out). Similar considerations would hold, for example, for the RFID tags attached to any product parts, which spend long times together, starting with the manufacturing process, and continuing throughout the product's lifetime.

One could consider an architecture in which the devices are prompted to initiate a key exchange protocol by a (human or machine) trusted party to facilitate an authenticated key exchange. In such a case, the authentication is provided through, and as such, it is subject to the security of the authentication of this trusted party – which in turn would have to rely on pre-programmed keys and would be subject to human error and leak of credentials. Software vulnerabilities due to human error have been well documented. Brubaker et al. [1] conducted a study to show various *incorrect* certificates, either due to expiration, corrupted authorities, or invalid data, would be accepted by one TLS implementation while getting rejected by another. Moreover, authors in [2] attributed the usage of insecure "reusable code" from Stack Overflow as a source of human error. In this paper, we construct a key establishment protocol that achieves authentication through the duration of co-location and works on the principles of Zero Configuration Networking [3].

The concept of *quality-time advantage* was introduced in [4] and further developed in [5]. It implies that, while the pairing devices are able to spend long and largely uninterrupted periods of time (referred to as *quality time*) in the proximity of each other, an attacker is limited in how much time she can spend in their vicinity. For our application, assume that the attacker can reach the devices' vicinity only after breaching the physical defenses of the storage facility. By the time the attacker can intercept – and interfere with – the communications between devices, the key establishment protocol is already well on its way – and it may continue well beyond the interval of time that the attacker can afford to spend at the location without being found. While not directly related to our large-scale, low-cost applications, a similar attack model is adequate for body-area networks – in this case, an attacker can only follow the intended victim for a short period of time, lest he is discovered (such an attacker is appropriately referred to as *"the man on the bus"* in [4]).

Co-located devices can establish a symmetric key between themselves through the application of *quality time*. Consider a scenario where a large number of military-grade wireless sensors are needed to be deployed (possibly in a complete loss manner), and before deployment, they are kept together in a storage facility. Consequently, these sensors can utilize the *quality time* they spend together to establish a secure channel among themselves. Similarly, think

Shahnewaz Karim Sakib and Yong Guan are with the Department of Electrical and Computer Engineering, Iowa State University, ssakib@iastate.edu, guan@iastate.edu.
George T Amariucai is with the Department of Computer Science, Kansas State University, amariucai@ksu.edu

of an industrial control application where a considerable number of sensors are deployed to ensure the automation of an industrial process and minimization of user intervention. These sensors spend most of their time together in a secure location, conceivably in a warehouse, and thus can make use of their shared time to establish a symmetric key to create a secure channel among themselves.

Let us consider the automobile industry as an example of large-scale production, as demonstrated by the production of more than 1000 vehicles an hour in North America in 2019 by Toyota [6]. An electronic control unit (ECU) takes input from several parts of a vehicle and makes a decision based on the collected information. Therefore, to ensure secure communication between separate vehicle parts, we need to establish a symmetric key. The cost of relying on PKI or trusted third parties to achieve this key can be overwhelming. The notion of quality time suits this situation perfectly, as the ECUs that end up on a certain vehicle can count on spending the rest of their lives together. They can easily establish a key among themselves at any one time or do so repeatedly, based on nothing but their quality time advantage.

To understand how much information about a secret can be gained over time, authors in [4] introduced the notion of *information transfer function*. An information transfer function is defined as a rate at which a legitimate device can have information about a secret. In *Shamir's secret sharing scheme* [7], threshold cryptography is used, and consequently, the legitimate listening party has no information about the secret even when he has only one clue less than the required threshold. Consequently, there is a sudden rise in the information instead of a smooth transition. In other words, *the information transfer function of Shamir's secret sharing scheme is a step function*. This result is undesirable, as it lacks the flexibility of customization.

A co-location-based key establishment protocol was developed in [5]. The authors' employed threshold cryptography – specifically *Shamir's secret sharing scheme* [7], and successfully met the main design requirements set forth in [5]. It does, however, suffer from certain drawbacks. For instance, the protocol – as well as the original design requirements – is designed to be secure solely against passive attackers. Moreover, due to the utilization of Shamir's secret sharing scheme, the resultant information transfer function is a step function. The issue is handled in [5] by running multiple instances of the protocol in parallel, resulting in significant overhead.

The current paper attempts to improve on the protocol of [5] by solving these issues. Specifically, the contribution is twofold: (1) we propose a new algorithm for co-location-based key establishment, which is robust against active attackers, and (2) we introduce a methodology to make the algorithm consistent with any arbitrary information transfer function [5] without using parallel instances; interestingly, this methodology is also applicable to the protocol of [5].

The remainder of the paper is organized as follows. Related works are discussed in section 2. In section 3, we analyze both the system and the attack model and provide definitions that formalize our attack model and the security and robustness of our protocol. Additionally, we introduce an evaluation framework against which we evaluate our proposed protocol. The *ZeroProKeS* algorithm is presented in section 4. Section 5 describes the error correction capability of the protocol. A security analysis is provided in section 6 and section 7 determines the computational cost of the protocol. Section 8 shows how the protocol can follow any arbitrary information transfer function. The implementation and the evaluation of the protocol are discussed in section 9, while section 10 presents the summary and concludes the paper.

2 Related Work

With the advent of RFID technology, various low-cost applications have been launched to perform secure key establishment. Elliptic curve cryptography (ECC) based key establishment protocol has been proposed to achieve low power consumption [8], whereas in [9], the authors combined this ECC with a symmetric key operation to design a dynamic key exchange protocol.

In a typical symmetric key cryptographic protocol, the communicating parties are assumed to have some kind of an advantage over the adversaries – for example, they may be communicating over an exceptional communication channel [10], [11], or they may have access to a source of common randomness, either relying on network metadata [12], [13] or obtained from manipulating auxiliary random sources in the physical layer [14], [15], [16].

In the IoT literature, *context-based pairing* has been introduced to eliminate the human involvement while pairing up IoT devices. One such technique is to extract entropy from the surrounding environment by converting common sensor measurements to common randomness, which provides the basis for symmetric key establishment [17], [18], [19]. The authors in [20] utilize a similar idea of harvesting randomness from the environment to implement an autonomous, secure pairing protocol named *Perceptio* for IoT devices.

Eschenauer et al. [21] introduced a probabilistic key distribution approach for symmetric key establishment. In the paper, the authors proposed a random selection and distribution of the key. The authors in [22] and [23] improve this method by reducing the memory space required for key storage and by improving the connectivity among the nodes of the network.

The idea of *Minimalist Cryptography* was introduced by Juels et al. [24]. It provides effective tracking protection but needs to exchange mappings when a product is transferred. Authors in [25], [26] suggested disabling the RFID tag once it has performed the assigned task to prevent unwanted affiliation. Such disabling of tags, however, limits the usage of RFID tags. Mathur et al. [27] devised a protocol for a key establishment that exploits the assumption that the channel response, which decorrelates quickly, is unique between two communicating entities. Various information-theoretic approaches have also been analyzed in this context. Authors in [28], [29] introduced the idea of generating the common bits based on the information that is available to both valid users but not fully available to the attacker. A number of protocols, such as [30], [31], [32], utilize this approach to establish the secret key.

Several password-authenticated key exchange protocols have been introduced in recent years [33], [34], [35]. Such

protocols are immune to offline attacks. Additionally, there have been numerous research on adopting PAKE protocols for the post-quantum world [36], [37].

In recent times, smart-card-based password authentication has been widely popular for two-factor authentication in a diverse application domain. Various cryptanalytic attacks, most notably side-channel attacks, have been launched to gain unauthenticated access to secrets [38], [39]. Therefore, considerable work has been done to design a secure two-factor authentication scheme. In [40], [41] authors discussed the security of such a two-factor authentication scheme based on the non-tamper resistant assumption of smart cards. Authors in [42] introduced an efficient authentication scheme for mobile devices that is quantum-resistant. Finally, Wang et al. [43] proposed a systematic evaluation framework that has been adopted in numerous recent works [44], [45], [46], [47] to evaluate the security of a smart-card-based authentication protocol.

The development of quantum computers opened the door for numerous key exchange protocols for the post-quantum world. Among them, a lattice-based cryptosystem is considered the most promising [48]. Ding et al. [49] introduced a reconciliation-based Ring-LWE key exchange scheme for two-factor authentication. The protocol is a variation of the PAKE protocols [50]. Several other reconciliation-based PAKE protocols were introduced in [51], [52].

The idea of time advantage was first introduced in the *adopted-pet (AP) protocol* of [4] for RFID applications. It deals with breakable stream ciphers to establish the key. The reader would query the tag, and the tag would reply with a small chunk of the key stream that it received when it accessed the cipher as a result of the query. Once the reader has gathered enough shares, it would be able to compute the secret as the characteristic polynomial. Therefore, with the correct amount of chunks, solving the characteristic polynomial reduces to solving a linear system of equations. Contrarily, an adversary may fail to gather such an amount of chunks, and thus, she would have to solve a system of quadratic equations over a finite field, which is known to be NP-complete [53]. However, *AP Protocol* releases the information about the secret at a linear rate, making the information transfer function of the protocol linear. But it is expected that the transfer function increases gradually with the length of uninterrupted time the reader spends with the tag so as to leak less information about the key.

This direction was further explored in the patent of [54]. The author achieved the key establishment by controlled release of the plaintext. One of the legitimate parties breaks the secret into various shares and later encodes these shares with an erasure code. Afterward, it releases the encoded-word corresponding to one of the shares over a certain interval of time. The device that listens to this broadcasting for a sufficient time interval will have the share of the key. Once it gets a large enough number of shares, it will be able to generate the secret key. However, for correct operation, one device needs to be aware of the beginning of the listening of other devices. This awareness can be achieved by making a query to the secret holding device, but this method paves the way for denial of service attacks. Moreover, because of the employment of *Shamir's secret sharing scheme* [7] to compute the shares, no information is obtained about the secret until the last share is computed. It makes the protocol very inflexible to changes.

The method of [54] was further improved in [5]. In that paper, the authors also utilize *Shamir's secret sharing scheme* to break the secret into various shares for the QTAB-KEP. However, in order to make it start time-independent, the scheme is run in parallel. The number of batches running in parallel is equal to the total number of clues. It ensures that the legitimate party can have the secret if she listens for a certain uninterrupted amount of time. This batching also has a nice property as it also enables the device to control its information transfer function, unlike the basic QTAB-KEP. However, this method is vulnerable to active injection if the number of injected clues overcomes the error correction properties of Reed-Solomon code [55].

3 The Quality-Time-Based Key Establishment Problem

3.1 Preliminaries

In this subsection, we shall briefly discuss the concept of co-location and how to identify the co-location. Consider the example of the deployment of a substantial number of low-cost sensors to monitor the air quality of a specific region. These sensors are closely located and monitor the same space for identical or different purposes. Moreover, these sensors communicate with each other to make a collaborative decision. These sensors, occupying a similar spatial region, are referred to as *co-located sensors*. It is possible that a monitoring device collects data from these *co-located sensors* and makes a decision based on the reasoning of all these co-located sensors. Such decision-making is known as collective intelligence. Kotevska et al. [56] utilized this collective intelligence to determine the behavior patterns among correlated sensors. Therefore, ensuring secure communication between these co-located sensors is of utmost importance.

Now we shall discuss how to detect the co-located sensors, more specifically, how to detect that several sensors have spent *quality time* together. Recall that *quality time* is defined as the size of an uninterrupted time interval that the two devices can spend together. We take the approach of [5] – namely, the quality of the time interval can be measured through the implementation of puzzles and emission of *clues* regarding the solution of the puzzle. The clue issuing device will generate a puzzle and, at regular/ irregular time intervals, release some *clues* regarding the solution of this puzzle. Each *clue* will have some information that will enable a device to solve the puzzle and, correspondingly, prove the co-location to the clue issuing device. A valid user, who will be able to listen to these clues with no or very few interruptions for a considerable amount of time, will be able to establish a symmetric key with the emitting device by proving its spatial closeness to the clue issuing device.

3.2 System and Attack Models

The limitation of extracting the secret key from the common randomness, enabled by a superior communication

or observation channel, is that the security of this type of protocol is conditioned on the superiority of the channel. Considering a planned uninterrupted time between the parties eliminates this restriction. An interesting thing to note here is that considering only the amount of time that the two legitimate parties spend in the company of each other cannot render the protocol secure, as an attacker who can spend many distinct small time intervals with one of the devices should be able to eventually accumulate the required amount of time. Therefore, a better criterion would be the *quality* time that two devices can share together. In the previous subsection, we have described how to recognize the *quality time*. Such a measure ensures the detection of co-located sensors. Yet, there may be missed or injected clues due to the nature of wireless communication. To formalize these cases, we formulate various types of access that a legitimate user or an attacker can have during the execution of our proposed quality-time-advantage-based protocol, referred to as *ZeroProKeS*.

Definition 1. Legitimate user access: We say that a legitimate user has (g, w) legitimate access to the protocol if he can listen to intervals of at least g consecutive clues, out of which he misses at most w clues.

Definition 2. Passive attacker access: We say that an attacker has (m, p) passive access to the protocol if she can listen to intervals of at most m consecutive clues and must miss at least p consecutive clues between two such listening intervals.

Definition 3. Active attacker access: We say that an attacker has (m, p, r) active access to the protocol if she can listen to intervals of at most m consecutive clues, must miss at least p consecutive clues between two such listening intervals, and can modify at most r arbitrary clues during each listening interval.

Based on these various types of access, we formulate the properties of *ZeroProKeS* along the lines of the three definitions below.

The first definition, introducing the $robustness$ criterion of the protocol, is similar to Definition 2.1. of [5].

Definition 4. Robustness: The protocol is said to be (g, w) robust if a legitimate party who has (g, w) legitimate access to the protocol can retrieve the secret with probability 1 in the absence of any active attackers.

This definition indicates that there should be at least w redundant clues among the g consecutive clues. There can be several ways to achieve this redundancy. The obvious choice is to employ some form of an error correction code.

Apart from being robust to the missed or erroneous clues, *ZeroProKeS* needs to be secure against an eavesdropper as well. As the eavesdropper spends less time in the proximity of the clue issuing device, she needs to be penalized for the long separation. Thus, we can formulate the following definition for the $security$ of the protocol – in essence, a probabilistic version of Definition 2.2 in [5].

Definition 5. Security: The protocol is said to be (m, p, δ) secure if an attacker with (m, p) passive access to the protocol can only recover the secret with probability less than δ.

The probabilistic formulation in the definition above allows us to move away from the threshold cryptography method of [5] and thus produce a more efficient protocol.

Finally, *ZeroProKeS* needs to be immune against the actions of an active attacker. An active attacker can inject or modify clues along with the passive listening. Therefore, we need to develop the $resilience$ of the protocol to circumvent an active attacker.

Definition 6. Resilience: The protocol is said to be (g, v, m, p, r) resilient if a legitimate party that has (g, v) legitimate access to the protocol can recover the secret with probability 1 in the presence of an attacker who has (m, p, r) active access to the protocol.

As the legitimate user has (g, v) legitimate access, they can miss at most v clues during their listening interval of g clues. Moreover, the active attacker has (m, p, r) active access and thus, can inject r clues during their listening interval of m clues, after which they must miss consecutive p clues.

This definition indicates the maximum number of clues that an attacker can modify in an arbitrary fashion such that the legitimate party is still able to recover the secret.

3.2.1 Security Assumption

In our proposed key establishment protocol, once a key is established between the sender and receiver, the receiver sends a message to the sender indicating that they have the correct key. We have assumed that such a message from the receiver to the sender can not be interfered with by the adversary. This is the only security assumption needed for our protocol to operate properly.

3.3 Evaluation Framework

In this section, we shall introduce the characteristics of the attacker, the usability requirement that we aspire our protocol to maintain, and the evaluation metrics that a quality-time-advantage-based key establishment protocol should aim to satisfy.

3.3.1 Characteristics of the Attacker

Now we shall describe the characteristics of our adversary. A widely accepted security model is proposed by [42], [43]. Such an adversary model was proposed for smart-card based password authentication. We adopt the characteristics of the adversary for our *quality-time* framework, and list the characteristics as follows.

- C1 We give the adversary full control over the communication channel during the time period for which the adversary can access the channel. During the listening period, the adversary is assumed to have the ability to monitor the communication or modify the communication.
- C2 The adversary can enumerate all the possibilities in the password space based on their collection of clues.

C3 It is possible for the adversary to gain the knowledge of the key of any of the previous sessions.

3.3.2 Usability Requirements

These requirements ensure that the protocol is functional for large-scale deployment, and a user not familiar with the intricacies of a protocol can still be able to use the protocol without any significant issues. Consequently, we adopt the usability benefits listed in [57] for a quality-time-advantage-based key establishment protocol.

U1 The devices engaged in the key establishment protocol usually do not have to remember any of the previously established keys. Due to the nature of the *quality-time-advantage*, the secret needs to evolve periodically. Therefore, such a memory allocation for remembering a secret is not desired.
U2 The protocol should be applicable to a large-scale application without increasing the burden on a device.
U3 It is expected that a device should not require any additional physical object, such as a piece of paper or a key, to conform to the protocol.
U4 The protocol should not require any physical user effort and should be automatic and stand-alone. If the user effort is limited, we shall refer to the protocol as *Quasi-Effortless*.
U5 The protocol should be flexible to allow a valid device to establish a key with the clue issuing device, even if the receiving device misses several clues.
U6 The protocol should correctly identify the valid user and not reject a genuine user even in the presence of an active adversary.

3.3.3 Evaluation Metrics

Finally, we shall present several evaluation metrics against which we shall compare our proposed protocols. Note that the metrics were introduced in [43] as evaluation criteria for smart-card verification, and therefore, some of the criteria related to smart cards do not apply to our time-advantage-based protocols. We adopted the rest of the metrics to appropriately reflect the evaluation criteria for quality-time-advantage-based protocols.

E1 Neither the sender nor the receiver should maintain a database of the previously used keys.
E2 The protocol should be resistant to any known attacks, such as online/offline password guessing attacks or key reuse attacks.
E3 A session key should be established between the sender and receiver after the completion of the protocol.
E4 The protocol should be start time-independent, and any valid user can establish a key with the sender if they follow the protocol properly.
E5 The sender and receiver should authenticate each other at the end of the protocol.
E6 The protocol should ensure forward secrecy. Therefore, even if the most recent key is compromised, the adversary should gain minimal information regarding the current session.

3.4 Problem Statement

With both the system and attack models and evaluation framework already explained, it is now time to formalize the problem statement. Following a similar design path as in [5], a secure key establishment protocol based on quality-time advantages should aspire to be:

1) (g, w)-robust. It is desirable that g is as small as possible, while w is as large as possible.
2) (m, p, δ)-secure, with m as large as possible, and p as small as possible for some given security parameter δ.
3) (g, v, m, p, r)-resilient, with v, m, r as large as possible, and g, p as small as possible. This property covers resistance to replay attacks, fake clue injection and clue jamming.
4) start-time independent, meaning that any legitimate party can start from any time and if he follows the protocol accordingly, he will be able to successfully learn the secret key.
5) completely automatic and stand-alone.

The first three requirements deal with the fact that a valid party may receive erroneous clues, miss some clues from the clue issuing device, or see some injected clues. The proposed protocol employs some redundancy, in the form of erasure codes, along with a separate mechanism for error detection to deal with these missing clues. This is preferable to a single error correction code because it yields better robustness and efficiency. The fourth property is required to ensure that the system is invulnerable against trivial denial of service attacks and leads to the clue-issuing device maintaining a single session for all potential legitimate parties. The last property renders the protocol independent of other security protocols and thus enables it to have context-free security guarantees [5].

4 The ZeroProKeS Key Establishment Protocol

4.1 ZeroProKeS Algorithm

4.1.1 Clue generation and coding across clues

We will refer to the clue issuing device as Alice, the legitimate listening party as Bob, and the attacker as Mallory. As a first step, Alice has to generate some random numbers and broadcast them as clues. Recall that *ZeroProKeS* has to be non-interactive, as feedback from Bob may render it susceptible to trivial denial of service (DoS) attacks. For generating the random numbers, we will assume that Alice is in possession of a true random number generator. Finally, the key could be a hash of the concatenation of these random numbers. In such a case, Bob will have no idea if the numbers he received are correct or erroneous. Additionally, Mallory could easily inject fake clues or modify a clue.

To learn the secret, Bob needs to know the indices of the clues that are erroneous or completely missing. To address this issue, after a certain number of clues, *ZeroProKeS* proposes that Alice transmits a separate list of hashes corresponding to those clues – these provide clue integrity verification. Once Bob has the list, he will compute the hash for all the previously received clues and perform a set intersection between his list of clues and the received list of hashes. This method has a few helpful properties.

If Bob receives any clue incorrectly or receives injected or modified clues, the hash of that clue would be different (with high probability), and it will not show up in the intersection. Additionally, suppose Bob receives a repeated clue, i.e., Mallory performs a reply attack. In that case, the hash of the clue would be the same, and the set intersection ensures that the hash is displayed only once.

Once Bob performs the set intersection and gets the list of correctly received clues, he will check to see if he has collected enough clues to find the secret. Therefore, the protocol needs to include some redundancy. For this purpose, we will employ an $(n, n-k)$ erasure coding [58]. The erasure coding scheme ensures that if Bob receives $(n-k)$ correct clues out of the n consecutive clues, then he would be able to obtain the rest of the k clues. Once Bob performs the intersection, he knows exactly which clues are correct and which are wrong. Using an erasure code rather than a simple error-correction code achieves a higher error correction rate – albeit at the cost of the additional overhead associated with the transmission of the hash list.

Alice will use an $(n, n-k)$ erasure code with generator matrix, $\mathbf{G}_1$. The elements of $\mathbf{G}_1$ are in the field $GF(2^{l_c})$, where l_c is the length (in bits) of the clue. Alice first generates $(n-k)$ random numbers, and afterwards uses the matrix $\mathbf{G}_1$, of size $((n-k) \times n)$, to generate the rest of k clues. The matrix $\mathbf{G}_1$ is in canonical form, so the first $(n-k)$ columns will represent an identity matrix and the last k columns will represent the weights to calculate the k redundant clues. If we represent this latter matrix as $\mathbf{A}$, then $\mathbf{G}_1 = \begin{bmatrix} \mathbf{I}_{(n-k)} & \mathbf{A}_{k\times(n-k)} \end{bmatrix}$. The matrix $\mathbf{G}_1$ can be either public knowledge, or it can be communicated by Alice, once in a while, as part of the protocol.

4.1.2 Blocks of clues and coding across blocks

So far, we have discussed how Bob will get the correct clues from Alice. It required a set intersection, computation of hashes, and employing erasure coding. However, Mallory can perform these operations as well. The security of *ZeroProKeS* relies on the expectation that Bob will spend more quality time with Alice, and therefore, Mallory should miss more than k clues out of every consecutive n clue. But there is also a possibility that Mallory can modify more than k clues and thus prevent Bob from learning the secret. Otherwise, Mallory could interfere with the list of hashes.

To address this problem, we provide an additional redundancy mechanism that extends beyond the reach r of Mallory. The entire clue stream is further divided into several blocks – each block corresponds to a codeword, along with an additional header and tail, as explained below – and the protocol ensures the correctness of clues for each block. The final secret can be computed to accommodate this new development by concatenating the clues from every block and then hashing the resulting string once more.

Now, to maintain the functionality of the protocol, even when Mallory jams the list of hashes meant for the identification of erroneous clues in each block, the protocol employs a linkage between the blocks, similar to the linkage between the clues within a block. Once Bob has a sufficient number of blocks, he can compute the rest of the blocks as well and subsequently produce the final secret.

For linking the blocks, the protocol again utilizes the erasure coding mechanism, with the generator matrix being $\mathbf{G}_2$, the elements of which belong to $GF(2^{(n-k)l_c})$. For this purpose, we will consider N blocks, and if Bob gathers the correct clues for at least $(N-K)$ blocks, among those N blocks, he will have the correct clues for the rest of the K blocks. The protocol will have a total of N_{tot} blocks. Alice will generate the clues for the first $(N-K)$ blocks, then use the $(n-k)$ independent clues of each of these blocks (so a total of $(N-K)$ strings of $(n-k)$ bits each) to generate the $(n-k)$ "independent" clues for each of the rest of the K blocks, using the matrix $\mathbf{G}_2$. Such generation of independent clues for K blocks is possible because of the utilization of $(N, N-K)$ erasure coding [58]. The matrix $\mathbf{G}_2$, in its canonical form, is a combination of identity matrix and some other matrix $\mathbf{B}$ and can be represented as $\begin{bmatrix} \mathbf{I}_{(N-K)} & \mathbf{B}_{K\times(N-K)} \end{bmatrix}$. Finally, for each of the new K blocks, Alice will use the matrix $\mathbf{G}_1$ to generate the rest of k dependent clues. Similar to $\mathbf{G}_1$, $\mathbf{G}_2$ can also be public knowledge, or can be sent periodically by Alice.

To further secure the protocol against active attackers, each block incorporates information from a selected number of previous blocks. Recall that an attacker can listen to at most m consecutive clues. Let us assume that m clues span a total of $(Z-1)$ blocks. We will include, in each block, some verification information regarding the previous Z blocks. The information regarding a block will be in the form of the hash of the clues in the block, in reverse order. For example, if the clues for the first block are $c_{1,1}, c_{2,1}, c_{3,1}$....., $c_{n,1}$, where the second subscript indicates the block number and the first subscript indicates the position of the clue in that certain block, then the verification information regarding this block will be encoded as $H(c_{n,1}||c_{(n-1),1}||.....||c_{1,1})$. The purpose of this mechanism is to enable Bob to recognize if the blocks that he has gathered previously are correct or not. To recognize the correct blocks, Bob will gather the clues of up to $(Z+1)$ blocks and then perform this hash computation for the previous Z blocks. Bob keeps verifying the correctness of the previous blocks until he accumulates a total of $(N-K)$ correct blocks. *It should be noted that* $(Z-1)$ *should be less than* $(N-K)$ *to ensure the security of the protocol.*

4.1.3 An instantiation of the ZeroProKeS algorithm

To summarize, *ZeroProKeS* establishes the key in two steps. The first step ensures that the clues in each of the blocks are correct; the second step, shown in Sub-Protocol 1, shows how to extend the previous algorithm to establish the secret. Note that in Sub-Protocol 1 we instantiated the clue length as $l_c = 512$ bits to provide a sense of the complexity of the algorithm in a real-world implementation.

We refer to the linking of clues in each of the blocks as the **inner-code** and the linking of the blocks as the **outer-code**. When we say the inner code is $(n, n-k)$, we mean that for each block, Bob needs to collect at least $(n-k)$ clues to recover all the clues of the block. Similarly to the inner code, if we consider the outer code to be $(N, N-K)$, it means that Bob can have all the clues corresponding to N consecutive blocks if he collects at least $n-k$ correct clues from each of $(N-K)$ blocks among the N.

Now we look at the structure of the block. Each of the blocks can be divided into three parts. The *Header* contains

Symbol	Meaning
n	Total no. of clues in inner-code
k	Redundancy in inner-code
N	Total no. of blocks in outer-code
K	Redundancy in outer-code
l_c	Length of the clue
N_{tot}	Total no. of blocks in the protocol
m	No. of consecutive clues an attacker can listen to
Z - 1	No. of blocks spanned by m clues
p	No. of clues attacker must miss after seeing m consec. clues
r	No. of injected clues by an attacker over the span of m clues
δ	Probability of attacker getting the secret

TABLE 1: Summary of notations

Z hashes: each hash is the digest of the concatenation of the clues, in reverse order, of one of the previous Z blocks. The *Body* contains the n clues, and the *Tail* consists of the list of hashes of the current blocks clues. Each clue is of length $l_c = 512$ bits, and there are in total n clues in each block. Now for computing the hash, we have various options. We can either use a secure cryptographic hash function such as SHA256 or a cyclic redundancy check (CRC) code such as CRC64. The output of SHA256 is 256 bits. Therefore, the length of the header would be $256Z$, and it would be equivalent to the length of $\frac{Z}{2}$ clues. Similarly, the length of the tail would be $256n$, which is the length of $\frac{n}{2}$ clues. So, using SHA256 will introduce an overhead of $\frac{n+Z}{2}$ clue lengths. Now, if we use CRC64, the header would be the length of $\frac{Z}{8}$ clues (i.e., $\frac{Zl_c}{8}$ bits), and the tail would be the length of $\frac{n}{8}$ clues ($\frac{nl_c}{8}$ bits). Thus the overhead will be reduced to $\frac{n+Z}{8}$ clue lengths.

We should note here that, for the protocol to function properly, we don't need secure cryptographic hash functions to compute the headers and tails of our blocks. In the worst-case scenario, Mallory can interfere with at most $(Z-1)$ consecutive blocks. However, as each block has the information about the previous Z blocks, Bob will be able to identify the correct blocks nevertheless. That is why we can allow a CRC64 implementation of the clue digests. It should also be noted that we don't need to worry about the overhead for computing the final secret, and thus, we can use a cryptographic hash function such as SHA256.

For simplifying the presentation, in the remainder of the paper, we shall refer exclusively to this instantiation of the protocol, with $l_c = 512$ and CRC64 clue digests. We call it the **512-64 protocol**. All the other parameters are kept in an alpha-numeric form. The summary of the notations used in the protocol can be found in Table 1.

4.2 Summary of ZeroProKeS Algorithm

This subsection summarizes the ZeroProKeS Algorithm and explains some steps of Sub-Protocol 1. The algorithm primarily works on two basic ideas: the linking between several clues inside a single block (referred to as **inner-code**) and the linking between several blocks themselves (referred to as **outer-code**). The inner code assures that if Bob has $n-k$ correct clues for each block, he can make use of these codes to get the rest k clues of that specific block. Likewise, the outer code ensures that if Bob collects $N-K$ blocks correctly, he can utilize these collected blocks to get the $n-k$ independent clues of the rest of the K blocks.

Sub-Protocol 1 Method for Establishing the Secret Key

- Bob collects up to $(Z+1)$ blocks and concatenates the clues in reverse order for each of the previous Z blocks to compute the hash of each block.
- The header of the $(Z+1)^{th}$ block contains those hashes of the previous Z blocks. Bob will compare the hash that he computed, to the header of the $(Z+1)^{th}$ block to identify any damaged blocks.
- As the outer-code is $(N, N-K)$, Bob checks to see if he has at least $(N-K)$ valid blocks among those $(Z+1)$ blocks. An important point to note here is that the maximum number of blocks Mallory can inject, over the span of N blocks, has to be less than $(N-K)$ in order to prevent her from interfering with the outer-code.
- Once Bob identifies $(N-K)$ valid blocks, he would use the $(n-k)$ independent clues of each of the valid blocks to compute the $(n-k)$ independent clues for each of the rest of the K blocks using the generator matrix $\mathbf{G}_2$.
- If Bob fails to verify $(N-K)$ valid blocks, he will collect an additional $(Z+1)$ blocks and try to verify $(N-K)$ valid blocks among these $2(Z+1)$ blocks.
- Subsequently, if Bob fails again to verify correct number of blocks, Bob will collect another $(Z+1)$ blocks, and this time he is guaranteed to be able to correctly identify $(N-K)$ valid blocks. Later, he will use the header of the correct blocks to determine rest of the correct blocks among the collected $(3(Z+1))$ blocks.
- Next, Bob will use $\mathbf{G}_1$ to compute the k additional clues for each of the K blocks.
- Finally, Bob computes the secret key as $key = H(c_{1,1}||c_{2,1}||.....||c_{n,N_{tot}})$, where $H(\cdot)$ may be implemented as SHA256 and $c_{n,t}$ indicates the n^{th} clue of the t^{th} block.

Moreover, the protocol also employs an additional linking between a certain number of blocks to secure the protocol against active attackers. Our attack model lets the adversary observe m consecutive clues encompassing $(Z-1)$ blocks. Once the adversary observes these $(Z-1)$ blocks, she misses p consecutive clues, which encompass at least $(Z+1)$ blocks.

At the beginning of the protocol, Bob will collect $(Z+1)$ blocks and then check if he has $(N-K)$ valid blocks among these collected $(Z+1)$ blocks. If not, he will repeat the process and check if there are $(N-K)$ valid blocks in the collected $2(Z+1)$ blocks. If Bob fails to obtain $(N-K)$ valid blocks again, he will repeat the process one final time, and this time he is guaranteed to have at least $(N-K)$ valid blocks among the collected $3(Z+1)$ blocks. Once Bob has obtained the $(N-K)$ correct blocks, he will utilize the linking (i.e., outer-code) between the blocks to find the $(n-k)$ independent clues of the rest of the K blocks. Once Bob generates $(n-k)$ independent clues for K blocks, he shall use the inner-code to generate the rest of the k dependent clues for each of those K blocks. Therefore, Bob will have each of the clues for all N blocks. Finally, he will hash them together to generate the secret. The flowchart of the summarized method is shown in Figure 2.

Now, note that we have mentioned that Bob is guar-

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3207927

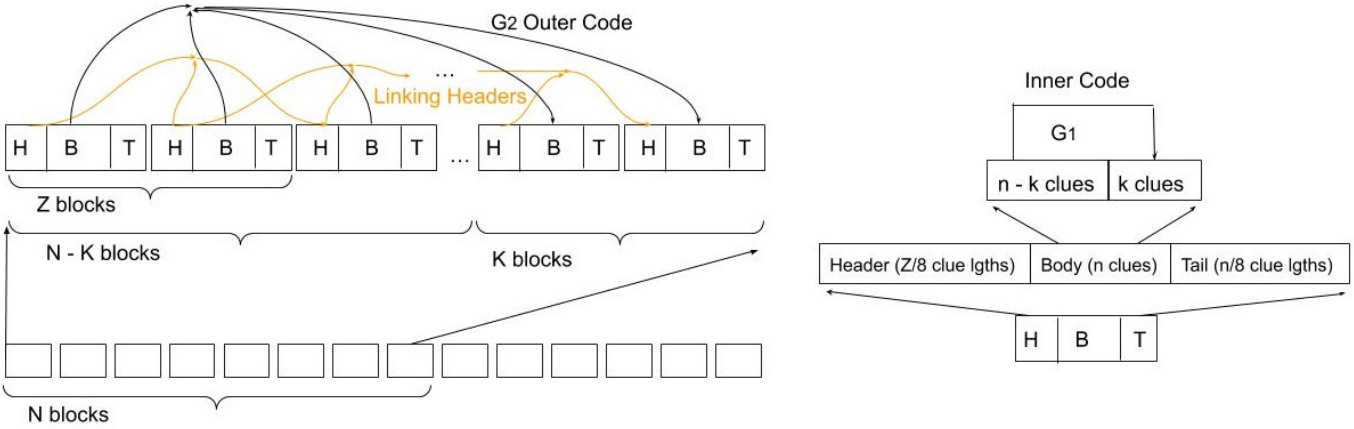

Fig. 1: Left: Block structure of the proposed 512-64 protocol; Right: The structure of a single block.

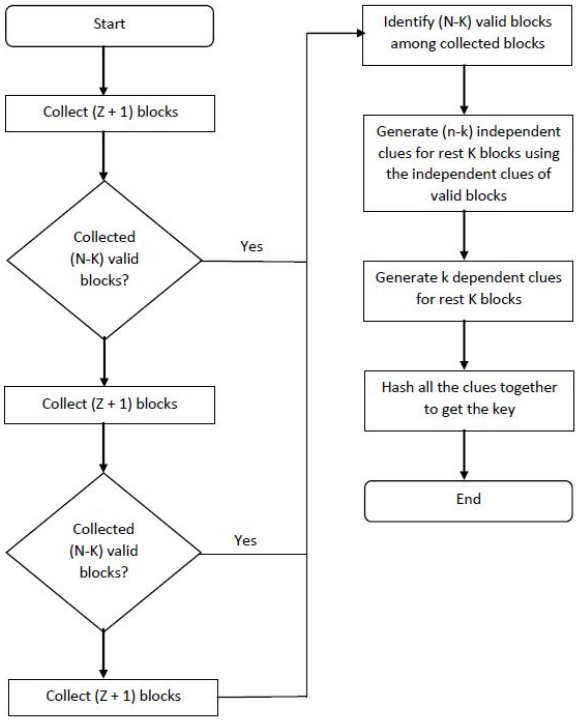

Fig. 2: Flowchart of the summarized ZeroProKeS algorithm

anteed to find at least $(N-K)$ valid blocks once he has collected $3(Z+1)$ blocks. We shall now demonstrate such reasoning with an example. Recall that to ensure the security of the protocol, $(Z-1)$ should be less than $(N-K)$ as indicated in subsection 4.1.2. If such a constraint is maintained, it is possible for Bob to recover $(N-K)$ valid blocks, even in the worst-case scenario.

For explanation, let us assume $(Z-1)=5$, $N=10$, and $K=2$. Initially, Bob collects $(Z+1)=7$ blocks. Let us assume among those collected 7 blocks, Mallory has injected the first 5 blocks. Therefore, Bob has 2 valid blocks ($6th$ and $7th$ respectively), which is less than $(N-K)=8$ blocks. After listening to 5 blocks, Mallory will miss $(Z+1)=$

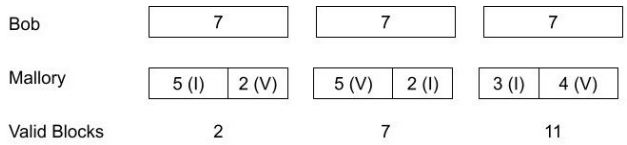

Fig. 3: Pictorial representation of the reasoning of why Bob is guaranteed to have $(N-K)$ valid blocks among $3(Z+1)$ blocks. I indicates the blocks injected by Mallory and V indicates the valid blocks that are not corrupted.

7 blocks. Thus, if Bob collects further 7 blocks, the first 5 blocks are valid, and Mallory can interfere with the last two blocks. Thus, once Bob collects $2(Z+1)=14$ blocks, he has 7 valid blocks, which is insufficient. Finally, when Bob collects further 7 blocks, at most, the first 3 blocks can be injected by Mallory. Therefore, the rest 4 blocks are intact, and consequently, Bob has 11 valid blocks, which is higher than $(N-K)=8$. This example shows why Bob is bound to have at least $(N-K)$ valid blocks among his collected $3(Z+1)$ blocks. The pictorial representation of the example is shown in Figure 3.

5 Robustness of the Protocol

5.1 Error Correction Ability

For each of the blocks, *ZeroProKeS* can detect any erroneous or incorrect clue and can retrieve all the clues from any subset of $(n-k)$ clues. The incorrect clues can be found by computing the intersection between the hash list and the hashes of the clues collected by Bob. Similar to the association of the clues within a block, the relation between the blocks can also be utilized to correct block errors.

The matrix $\mathbf{G}_1$ has size $((n-k)\times n)$ and rank $(n-k)$. The corresponding parity check matrix, $\mathbf{H}_1$ has size $k\times n$ and rank k. Therefore, any $k\times k$ sub-matrix of the matrix $\mathbf{H}_1$ is invertible. Take a codeword C (of size $1\times n$) and let C' represent a shorter version of C, produced by removing the incorrectly received (or missing) clues from C. If $\mathbf{H}_1'$ is obtained from $\mathbf{H}_1$ by removing the columns corresponding to the missing components, then the resulting syndrome

$s' = C'[\mathbf{H}_1']^T$ is nonzero. If the missing components are C'' (of size $1 \times k''$, with $k'' \leq k$), and $\mathbf{H}_1''$ (of size $k \times k''$) represents the columns of $\mathbf{H}_1$ corresponding to the missing components, then we can also write $C''[\mathbf{H}_1'']^T = s'$. As the rank of $\mathbf{H}_1$ is k, any k'' subset of the rows of $\mathbf{H}_1''$ can be used to produce an invertible matrix of size $k'' \times k''$. The inverse of this matrix can be used to recover C'' from s'. It should be noted that no matrix inversion is necessary in practice – rather, a simple LU decomposition should suffice.

Similarly, the matrix $\mathbf{G}_2$ has rank $(N - K)$ and the rank of corresponding parity check matrix, $\mathbf{H}_2$, is K. Using a similar approach to the one demonstrated above, we can find the independent clues for all N blocks by utilizing the independent clues of at least $(N - K)$ blocks.

The following theorem quantifies the robustness of the protocol. The block structure for the proposed 512-64 protocol is shown in Figure 1, from where we note that each block is the length of $\frac{9n+Z}{8}$ clues. In Figure 1, the orange-colored arrow indicates the linking between the headers of the blocks, and the black-colored arrow indicates the utilization of inner and outer code.

Theorem 1. Given (N_{tot}, N, K, n, k, Z), the proposed 512-64 protocol is (g, w)-robust, where $g = N_{tot}\frac{9n+Z}{8}$ and $w = (K+1)((k+1)/8+1) - 1$.

Proof. We have a total number of N_{tot} blocks, and each block is the length of $\frac{9n+Z}{8}$ clues. Thus the total number of clues transmitted during Bob's allowed listening time is $g = N_{tot} \times \frac{9n+Z}{8}$.

For Bob to fail to establish the secret key with Alice, he must fail to receive at least $(K+1)$ blocks out of a codeword of N blocks. Recall that the outer codewords contain N blocks, and there is a total of $\frac{N_{tot}}{N}$ outer codewords available during Bob's allowed listening time. Missing any single outer codeword results in incomplete keying information.

There are various ways in which Bob can miss the blocks in a codeword. He can miss $k+1$ clues per block, or he can miss the parts of the tail of the block that corresponds to these $k+1$ clues (a tail is the length of $n/8$ clues, so this part of the tail would span the length of $(k+1)/8$ clues). If Bob misses part of the tail of a block, he is still able to recognize a correct block body by using the header of the blocks that follow it. However, this capability vanishes if Bob misses only one additional clue from the block's body. So, in total, Bob must miss $(k+1)/8+1$ clues of the block. To summarize, the minimum number of clues that must be missed to invalidate an outer codeword is $(K+1)((k+1)/8+1)$. Any fewer missed clues and Bob will be able to correctly recover the entire secret. □

6 Security Analysis

6.1 Attack Taxonomy

Based on the structure of a block, as shown in Figure 1, the *attack taxonomy* of the protocol can be represented as follows:

1) Secret key leakage: attacker listens passively according to the allowed access pattern and reconstructs the established secret key.
2) DoS attacks
   a) The protocol does not complete successfully – this takes modifying (by jamming or inserting) $K+1$ blocks out of any one outer codeword of N blocks.
   b) The protocol completes with mismatched secrets.

The protocol's immunity against *secret key leakage attacks* is formalized in Definition 5, while its immunity against both types of DoS atacks is formalized in Definition 6. In the remainder of this section, we discuss the active attacks and derive two theorems that further enable us to choose the design parameters for the ZeroProKeS protocol.

6.2 Protocol Completion in the Presence of Active Attackers

This subsection deals with the ways in which Mallory could attempt to prevent the completion of the *ZerProKeS* protocol.

6.2.1 The Effect of Block Injection

We shall first look into the case in which Mallory modifies the content of blocks transmitted by Alice to Bob (by *injecting a block* we mean overwriting a block transmitted by Alice). Note here that from Mallory's perspective, listening to the channel before injecting a block does have some benefits – if Mallory wants to insert a block that fits in the sequence of previous blocks, she needs to know the body of some of the previous blocks, so that she can include them in the header of the inserted block.

Bob can begin to verify the validity of the previous blocks if and only if the previous Z blocks and the current block are all received correctly (up to small errors that are correctable at the block level by the inner code). This implies that Bob needs to observe at least $Z+1$ correct consecutive blocks. Once the verification process is started in this manner, it can then be further bootstrapped to verify the validity of any blocks preceding this sequence of $Z+1$ blocks.

Let us assume the value of the maximum number of blocks Mallory can inject during a single listening interval is $(Z-1) = 15$. Thus, each block has a hash list of the previous $Z = 15+1 = 16$ blocks. This means that the header of the ith block contains $H(c_{n,(i-1)}||c_{n-1,(i-1)}||.....c_{1,(i-1)})$, ..., $H(c_{n,(i-16)}||c_{n-1,(i-16)}||.....c_{1,(i-16)})$, where $c_{k,i}$ stands for the kth clue of block i. Bob needs to observe at least $(Z+1) = 17$ correct consecutive blocks. Therefore, once Bob has access to the 18th block, he can check and later verify the previous 16 blocks.

We now consider the following scenario: Mallory missed the first 12 blocks, then listened to 4 blocks, and then injected 7 more blocks. After that, she let 2 valid blocks pass and then injected 2 blocks again before losing access to the channel. This is represented by Figure 4. Here, V represents valid blocks that Mallory allowed to be transmitted without modification, and I represents injected (or modified) blocks.

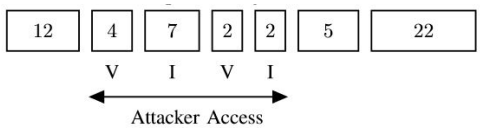

Fig. 4: Attacking header after listening to some blocks

When Bob gets the 18th block, he tries to verify the previous 16 blocks and fails. The verification procedure is that once Bob has 18 consecutive blocks, he compares the header of the 18th block with the hash list of the previous 16 blocks that he prepared. If they are the same, Bob labels those 16 blocks as valid. Note that even though Mallory injected the block after listening to previous 4 blocks, she does not have any information about the 12 blocks that preceded those 4 blocks. She can never get that information because if she decides to listen to 15 consecutive blocks, she will not have time to modify any more blocks. This argument also addresses the potential completion of the protocol with mismatched secret keys: such an event is not possible – the protocol either fails or completes correctly.

When Bob fails to verify the previous 16 blocks for the first time, he collects the next 18 blocks, tries to verify again in the same manner, and fails again. Next, Bob collects another 18 blocks and tries to verify them as well. This time, he is successful. Subsequently, using the header of those verified blocks, he is able to then verify all the rest of the valid blocks in the system. In Figure 4, Bob will at first verify the last 18 blocks, and then, using the header of those last 18 blocks, Bob will be able to verify the 5 blocks that he collected immediately after Mallory left the channel. Afterward, using the header of these 5 blocks, he will be able to verify the 2 blocks that are between the injected blocks. Similarly, using this back propagation approach, he would be able to verify all the valid blocks in the sequence. Even if Mallory injects the full 15 blocks, Bob will still be able to verify all the valid blocks.

It is important to note that if we used the hash list of the previous 15 (i.e., $(Z-1)$) blocks in the header instead of 16 (i.e., Z), Bob would be able to recognize the injected blocks, but he wouldn't be able to make sure that the initial 16 blocks were valid as well. This is demonstrated in Figure 5. If the protocol used 15 block hashes in the header, Bob would check every 17th block. In this case, once Bob had access to the 17th block, he would fail to verify the previous 15 blocks. He would fail to verify previous 15 blocks again once he reached the 34th block. Finally, when Bob reached the 51st block, he would be able to verify the previous 15 blocks. Using the header of those verified blocks, he would be able to verify the 2 blocks after the injected blocks, and using those 2 blocks; Bob would recognize the injected 15 blocks. However, as each block has the block hash of the previous 15 blocks, there would not be any link between the verified 2 blocks that are after the injected blocks and the valid 15 blocks that are sent prior to the interference of Mallory.

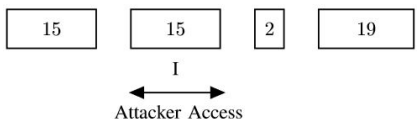

Fig. 5: Limitation of using $(Z-1)$ block hashes in header

In terms of our parameters, this means that the number of clues p that the attacker has to miss between observations must cover at least $Z+1$ blocks, or, for our protocol instantiation,

$$p \geq (Z+1)\frac{9n+Z}{8}. \quad (1)$$

Taking this together with the definition of m:

$$m = (Z-1)\frac{9n+Z}{8}, \quad (2)$$

we obtain the following two equations that, given the attacker's description by parameters (m,p), can be used to calculate the protocol parameters Z, n:

$$Z \geq \frac{p+m}{p-m}, \quad (3)$$

$$n = \frac{8m}{9(Z-1)} - \frac{Z}{9}, \quad (4)$$

with the observation that the protocol requires $p \geq m+1$.

6.2.2 What it Takes to Inject a Block

The body of a block consists of the clues emitted by Alice during the transmission of that block. Mallory has several ways in which she may attempt to distort the block.

Repeated clues: Mallory may replay a clue that was previously transmitted. As established in the description of the protocol, once Bob gathers the body, he will compute the hash of these clues and then perform a set intersection. If there are repeated clues, the set intersection will return only their first appearance.

Jammed /arbitrarily modified clues: Mallory chooses to jam either a part or the whole body, thus arbitrarily modifying the transmitted clues. If the number of jammed clues is less than k, Bob will be able to identify the jammed clues and recover the rest of the body with the help of the inner-code. Thus, the block is only compromised when at least $k+1$ clues are successfully jammed.

Jammed block tail: Mallory chooses to simply jam the block's tail, which contains the hashes of all clues in the block. To achieve any damage, Mallory has to jam at least $k+1$ of the clue hashes in the tail. This may appear to be the most efficient way of "modifying" a block, as, without the tail, Bob would not be able to recognize which clues are valid. Interestingly, however, once enough new blocks are gathered, the valid clues of the affected block can be identified, as illustrated above, using the headers of the new blocks, hence neutralizing the effect of the attack. The whole block is thus not entirely lost, and Bob need not rely on the outer code to recover it.

However, jamming a single additional clue of the block has the effect of creating a hash mismatch with the headers of the following blocks – recall that the hashes in the headers correspond to entire blocks, not to individual clues. To conclude, the single most effective way for Mallory to modify an entire block to the point where it needs to be recovered using the outer code is to jam/modify $k+1$ clue hashes from the block's tail, along with any single one of the block's clues – a total of $\frac{(k+1)}{8}+1$ clue lengths, as described in Theorem 1.

6.2.3 Comparison between Protocols

The comparison between *ZeroProKeS* and various other protocols, such as *QTAB-KEP*, *Diffie-Hellman Protocol*, *Key Wrapping*, and *Common Randomness Extraction*, is presented in Table 2. From the comparative analysis, we can see that *Diffie-Hellman Protocol* is not (m,p,δ) secure and *Common*

Randomness Extraction is not start time independent. Moreover, neither of them are (g, v, m, p, r) resilient. Even though *Key Wrapping* can be (g, v, m, p, r) resilient, it is not start time independent. Note that an argument can be made that the Diffie-Hellman key agreement can resist an attacker from breaking the protocol even if she gets the total message. Such an adversary framework is not compatible with the concept of the quality-time-advantage key establishment protocol, as the adversary is bound to miss p consecutive clues after observing m consecutive clues.

Protocol	Authenticated	Start Time Indep.	(m, p, δ) secure	(g, v, m, p, r) resilient
ZeroProKeS	Co-location	✓	✓	✓
QTAB-KEP [5]	Co-location	✓	✓	×
Diffie Hellman [59]	PKI	✓	×	×
Common Randomness [14]	Co-location	×	✓	×
Key Wrapping [60]	Co-location	×	✓	✓

TABLE 2: Comparison between protocols

6.3 Parameter Selection

From Theorem 1, we know that the total number of clues is $N_{tot} \times \frac{9n+Z}{8}$. Mallory has (m, p, r) active access, and during this time she can listen to $(Z-1)$ blocks. This can be represented by the Figure 6.

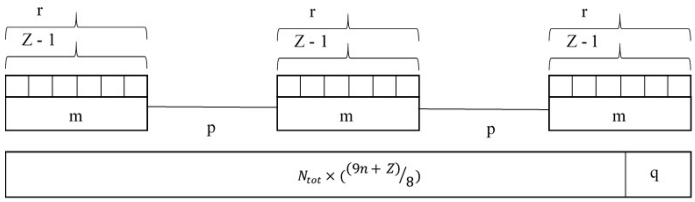

Fig. 6: Diagram for attacking model representation

From Figure 6, we see that over the span of $am+(a-1)p$ clues, the attacker can listen to at most am clues. The integer a can be found from the following two equations:

$$am + (a-1)p \leq (N_{tot} \times \tfrac{9n+Z}{8}) \tag{5}$$

$$(a+1)m + ap > (N_{tot} \times \tfrac{9n+Z}{8}), \tag{6}$$

or equivalently:

$$a \leq \frac{(N_{tot} \times \frac{9n+Z}{8}) + p}{(m+p)} \tag{7}$$

$$a > \frac{(N_{tot} \times \frac{9n+Z}{8}) + p}{(m+p)} - 1. \tag{8}$$

Theorem 2. Given (g, w, m, p, r), if a is given by (7) and (8), then a (g, w) robust ZeroProKeS protocol is also (g, v, m, p, r) resilient, where $v = w - ar - \min\{r, m-q\}$ and $q = \min\{m, (a+1)m + ap - g\}$.

Proof. We know $g = N_{tot} \times \frac{9n+Z}{8}$ and the value of n and Z can be computed from (3) and (4), using m and p.

Now, if we look at Figure 6, we can see that Mallory can inject r clues over the span of m clues. Therefore, over the a sequences of m consecutive clues seen by Mallory, she can inject at least ar clues. However, the maximum number of clues that Mallory can inject is a bit larger, and is given by $ar + \min[r, m-q]$, where $q = \min[m, (a+1)m + ap - g]$ (see Figure 6). These injected clues decrease the number of additional clues that may be missed by Bob due to non-malicious reasons, such as interference. Hence, the resilience of the protocol is now $v = w - ar - \min[r, m-q]$. □

Theorem 3. Given (m, p, n, k, N_{tot}), if a is given by (7) and (8), the probability that an attacker can recover the correct secret key from the 512-64 ZeroProKeS protocol is $\delta \leq \max\left[2^{-(N_{tot}(n-k)-L)l_c}, 2^{-448}\right]$, where $L = a(Z-1)(n-k) + \lceil \frac{m-q}{\frac{9n+Z}{8}} \rceil (n-k)$.

Proof. We first note that finding a clue from its CRC-64-based digest included in the blocks' headers involves a brute-force effort with complexity in the order of $2^{512-64} = 2^{448}$. This is for a single clue, but since the complexity is overwhelming, we choose not to consider this case at all – rather, we include the term 2^{-448} in the maximum above.

Now, without inverting CRC-64 digests, we can focus exclusively on eavesdropping information-bearing clues. We know that each clue consists of l_c bits. As each block has $(n-k)$ information-bearing clues, the total number of information-bearing clues that Mallory can listen to, over the a sequences of m consecutive clues, is $a(Z-1)(n-k)$. Additionally, she can access $m-q$ more clues – see Figure 6. As each block is of length $\frac{9n+Z}{8}$ clues, the additional number of full blocks that she has access to is $\lfloor \frac{m-q}{\frac{9n+Z}{8}} \rfloor$. Each of these blocks has $(n-k)$ information-bearing clues, so Mallory can eavesdrop an additional $\lfloor \frac{m-q}{\frac{9n+Z}{8}} \rfloor (n-k)$ information-bearing clues. Finally, out of the remaining clues (which do not cover a full block), Mallory can only access fewer than $(n-k)$ information-bearing clues. Consequently, in total she can access fewer than $L = a(Z-1)(n-k) + \lceil \frac{m-q}{\frac{9n+Z}{8}} \rceil (n-k)$. The total number of information-bearing clues is $N_{tot}(n-k)$, and so the probability of guessing all information-bearing clues correctly is less than $\frac{2^{Ll_c}}{2^{N_{tot}(n-k)l_c}} = 2^{-(N_{tot}(n-k)-L)l_c}$. □

Since for our 512-64 protocol $l_c = 512$, it should suffice to have $N_{tot}(n-k) > L$, or equivalently:

$$N_{tot} > a(Z-1) + \lceil \tfrac{m-q}{\frac{9n+Z}{8}} \rceil. \tag{9}$$

7 Computational Cost

The cost for Alice arises from the matrix multiplication. She has to perform matrix multiplication both over $GF(2^{l_c})$ and $GF(2^{(n-k)l_c})$. For *coding across clues*, Alice has to do multiplications over $GF(2^{l_c})$. The multiplication in $GF(2^{l_c})$ has a complexity of $\mathcal{O}({l_c}^2)$ [61]. However, using an efficient algorithm, such as *Karatsuba's algorithm* [62], this can be reduced to $\mathcal{O}({l_c}^{1.585})$. As Alice does this multiplication $k(n-k)$ times (the generator matrix is in canonical form, so the first elements of the codeword coincide with the independent clues), the complexity can now be written as

$\mathcal{O}(k(n-k){l_c}^{1.585}) = \mathcal{O}(nk{l_c}^{1.585})$. Moreover, Alice has to perform $(n-k)(k-1)$ additions over the same field as well and the addition in the field has a complexity of $\mathcal{O}(l_c)$ [61]. Thus, the complexity of Alice for *coding across clues* can be written as : $\mathcal{O}(nk{l_c}^{1.585}) + \mathcal{O}(nkl_c) = \mathcal{O}(nk{l_c}^{1.585})$.

For *coding across blocks*, Alice has to do $K(N-K)$ multiplications and $(N-K)(K-1)$ additions in the field of $GF(2^{(n-k)l_c})$. Using a similar approach, we see that this complexity reduces to $\mathcal{O}(NK(l_c(n-k))^{1.585})$.

Now, we analyze the complexity of Bob's task. For this, we will consider the parity check matrix, $\mathbf{H}$, corresponding to the generator matrices. For *decoding across clues*, Bob needs to first find the syndrome s' of the codeword, which is of size at most $(1 \times k)$, formed by the clues he gathered. For this, he needs to do $k(n-k)$ multiplications and $k(n-k-1)$ additions in $GF(2^{l_c})$. This has the complexity of $\mathcal{O}(nk{l_c}^{1.585})$ + $\mathcal{O}(nkl_c) = \mathcal{O}(nk{l_c}^{1.585})$.

Next, Bob needs to solve for C'' from the equation $C''\mathbf{H''}^T = s'$. Using LU decomposition, this will take at most $k^3/3 + k^2$ multiplications, and as many additions over $GF(2^{l_c})$. The complexity of the computation is thus $\mathcal{O}(k^3{l_c}^{1.585})$. Using a similar analysis, it can be shown that for *decoding across blocks*, the complexity for Bob's task is at most $\mathcal{O}(K^3(l_c(n-k))^{1.585})$.

Thus, in the worst-case scenario (for computational effort) when Bob has to correct $k-1$ missed clues out of each one of $N-K$ blocks in each outer codeword (for a total of $\frac{N_{tot}}{N}$ outer codewords), and then has to determine the remaining K blocks of each outer codeword, the overall complexity can be written as $\mathcal{O}\left(\frac{N_{tot}}{N}\left[(N-K)(nk{l_c}^{1.585}) + (K^3(l_c(n-k))^{1.585})\right]\right)$.

In [5], the authors needed to run the basic *QTAB-KEP* in parallel to make it start time-independent. If we apply our framework to [5] and compare the computational cost, we can see that for Alice, *ZeroProKeS* performs better than *QTAB-KEP* whereas, for Bob, the costs are comparable when $K \ll N_{tot}$. The details of the comparison are given in Table 3. Note that Table 3 compares the cost of the protocols that are start-time independent, and (m, p, δ) secure.

Computational Cost	ZeroProKeS	QTAB-KEP [5]
Alice	$\mathcal{O}(NK(l_c(n-k))^{1.585})$ + $\mathcal{O}(nk{l_c}^{1.585})$	$\mathcal{O}(N_{tot} \times \frac{9n+Z}{8})^3$
Bob	$\mathcal{O}(\frac{N_{tot}}{N}[(N-K)(nk{l_c}^{1.585})$ $+K^3(l_c(n-k)^{1.585})])$	$\mathcal{O}(N_{tot} \times \frac{9n+Z}{8})^2$

TABLE 3: Comparison of computational cost between protocols that are start-time independent and (m, p, δ) secure.

8 Controlling the Information Transfer Function

The information transfer function controls how much information is leaked, about the secret, over time. This information transfer function should be customizable to allow flexibility for the user. Suppose a user wants to establish a symmetric key between his car and the car key. In that case, he may wish the protocol to leak more information about the secret key while the vehicle is running. The attacker is more likely to have a smaller listening interval in this case than when the car is stationary.

Therefore, the protocol is expected to emulate any monotonic increasing transfer function. In [5], this was achieved by executing multiple instances of the protocol in parallel, each instance consisting of a different-length codeword. This approach suffers from significant overhead but cannot be avoided because the basic protocol of [5] communicates all the information at once (when the $(n-k)$th clue becomes available). By contrast, our proposed protocol delivers information in small increments as each clue leaks a small amount of information. The challenge is thus how to make this amount of information count more or less, depending on the desired information transfer function.

First, we need to understand how information transfer is achieved for each block. The inner code for the protocol is $(n, n-k)$. So, if Bob has $(n-k)$ correct clues, he can have access to all the correct n clues. Therefore, the information increases over the duration of the first $n-k$ clues and then remains constant over the next k clues, as can be seen from the *red* line in Figure 7.

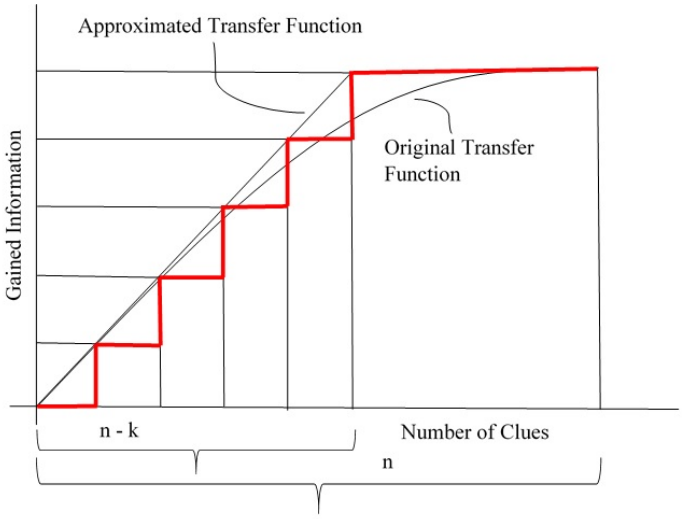

Fig. 7: Information transfer function for each block

Now the trick is to make different blocks convey more information. This is not straightforward, as due to the start-time independence requirement, all blocks have to look exactly the same. The solution is to assign different weights to different blocks a posteriori. That is, once Bob signals Alice that he has gathered enough clues, Alice and Bob both look back at the gathered blocks and assign different information gains to different blocks, according to the specified information transfer function. This is reflected in the production of the secret key by using a two-level hashing mechanism. For example, if the last block is supposed to convey less information than the second-last block, then the clues in the last block will be hashed into fewer bits than the clues in the second-last block. The results of all block hashes will then be concatenated and hashed again into a number of bits equal to the desired key length.

An implementation of this paradigm is illustrated in Figure 8. Notice that the whole information transfer function is sliced vertically into $N-K$ equal-width slices (in Figure

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3207927

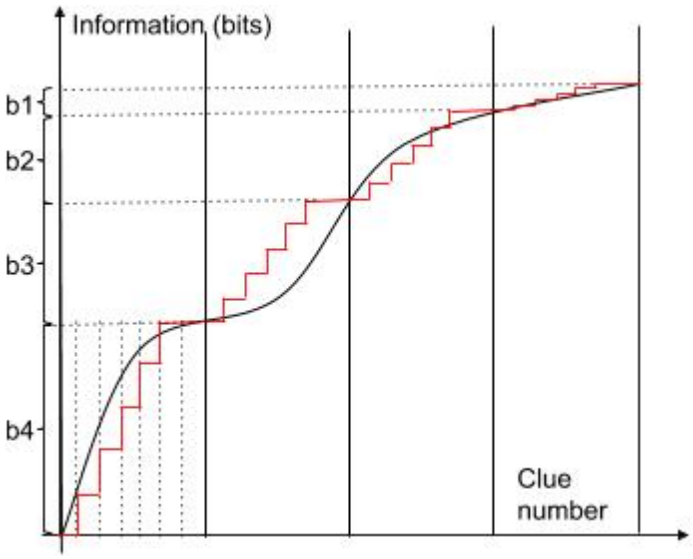

Fig. 8: Information transfer function for whole protocol

8, $N - K = 4$). Each slice is associated with a block and is further divided into n thinner sub-slices representing the clues in each block (in Figure 8, $n = 7$). Each slice is assigned an integer number proportional to the information transfer function's increase over the slice and represents the number of bits to which the clues in the corresponding block are going to be hashed in the first stage of the hashing process (in Figure 8, these integers are denoted as b_1, b_2, b_3, b_4). Notice that, over each slice, the information transfer function will be approximated by a stepped slope of $(n - k)$ equal steps, followed by a constant segment of length equal to the length of k steps, similar to the red curve of Figure 7 (in Figure 8, we have $n = 7$ and $k = 2$). By increasing the number of blocks in the protocol – i.e., by transmitting more frequent clues – we can approximate the information transfer function in this manner with arbitrary precision.

9 Experimental Results and Discussion

We implemented our proposed protocol using a Raspberry pi [63] as Alice and simulated an Android application, in Android Studio [64], as Bob. Our instantiation of *ZeroProKeS* uses the values $N = 16$, $K = 8$, $n = 16$, $k = 4$, $Z = 8$. If $N_{tot} = 32$, then for this example we have $g = 608$, $w = 17$, $m = 133$, and $p \geq 171$.

9.1 Performance Measurements

A suitable measurement quantity for such a low-cost protocol is power consumption. For measuring power, we have used the USB voltage, and current meter [65]. On average, the protocol draws an additional $49.56\ mW$ power (with an standard deviation of $0.1813\ mW$) above the baseline benchmark of $2\ W$. Thus, only an additional $2.478\ \%$ of the benchmark power is consumed by the protocol.

Another appropriate quantity to measure would be the average load on the processor while executing the protocol. The average load on the processor is defined as the ratio of the total computation time to the execution time of the protocol. As execution time is the parameter of the protocol, we have varied it to measure the effect of the execution of the protocol on the processor. The details of the average computation time for Alice and Bob can be found in Table 4. For computing over Galois field, we have used the implementation provided by Bouncy Castle [66].

	Operation	Sample Mean (μs)	95% Confidence Interval (μs)
Alice	Creating Header	0.97	[0.75, 1.19]
	Creating Body	20.9	[16.65, 25.15]
	Creating Tail	35.42	[25.57,46.27]
Bob	Verifying Header	128.40	[104.13,152.66]
	Verifying Body	42.00	[34.51,49.49]
	Verifying Tail	52.36	[44.79, 59.93]

TABLE 4: Average computation time

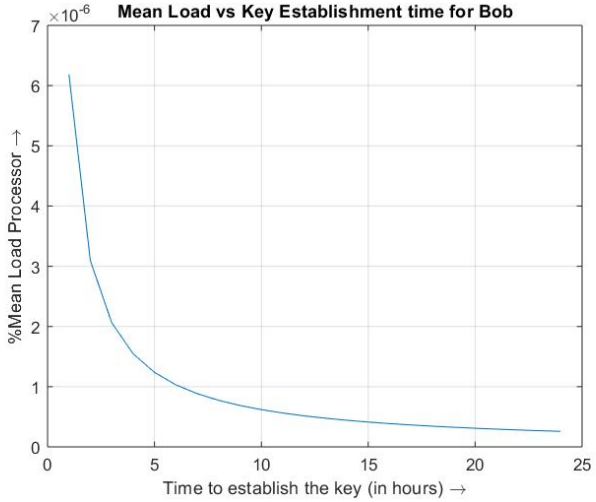

Fig. 9: % Mean load vs execution time for Bob

For each block, Alice generates $(n-k)$ random numbers and then uses $\mathbf{G}_1$ to generate the rest of the k numbers. Along with generating these numbers, Alice also computes the CRC-64 hash of the concatenation of these n random numbers in reverse order. The computational time to perform these operations is captured by the operation "Creating Body". "Creating Tail" simply indicates the computational time required to generate the CRC-64 hash of each of the numbers separately. And finally, the computational time to add the list of the hash of the concatenation of clues in reverse order for previous Z blocks is shown in "Creating Header".

For "Verifying Tail", Bob collects the clues, computes the CRC64 hash of each clue, and performs set intersection. "Verifying Body" captures the computational time for the generation of n clues for each block. We have considered the worst-case where Bob needs to compute k erasures from the correct $(n - k)$ clues.

For "Verifying Header", Bob has to perform three operations (in the worst-case scenario). At first, Bob needs to identify the correct $(N - K)$ blocks among all the collected blocks. After that, he needs to use $\mathbf{G}_2$ to reproduce the correct "independent" clues of these N blocks. And finally, using these "independent" clues and $\mathbf{G}_1$, Bob will need to find the dependent clues for all blocks.

Now we analyze the load, that is introduced on the processor, by execution of the protocol. Figure 9 shows the increase in the load percentage while executing Bob. Similar graph is obtained for Alice as well. The plot follows the exponential equation $ae^{bx} + ce^{dx}$. In this specific case, the values are : $a = 7.369 \times 10^{-8}, b = -0.4288, c = 6.727 \times 10^{-9}, d = -0.05162$.

It is apparent from Figure 9 that there is a trade-off between the time required for key establishment and the computation load. Note that "the time to establish the key" is a parameter of the clue-issuing device for the time-advantage-based protocols. The co-located devices are guaranteed to spend longer uninterrupted time together, while an adversary may gain access to such an environment after breaching the defense of the facility. Moreover, the attacker can not afford to have uninterrupted access to the facility without being found. Our protocol can exploit such a long time frame to establish a secure key among co-located devices while ensuring the nominal increase in the power usage for the device while implementing the protocol, as demonstrated by Figure 9.

9.2 Comparative Evaluation

9.2.1 Comparison against Adversary Characteristics

Now we shall compare several protocols against the characteristics of the adversary that were introduced in 3.3.1. Characteristic C1 portrays an active adversary who can interfere with the communication between Alice and Bob. We know that QTAB-KEP is not secure against such an adversary, whereas both the primitive Diffie Hellman and common randomness algorithms are susceptible to man-in-the-middle attacks. Moreover, the C2 characteristic illustrates an adversary who has a significant resource to enumerate all possible secrets based on her collection of clues. None of the protocols are susceptible to such an adversary. Finally, the final characteristic of the adversary ensures the compromising of the current session of the protocol if she gains knowledge of the previous session key. However, such an attacker can have access to the secret for key wrapping-based protocols. The summary of the result is shown in Table 5. Note that the ✓ sign indicates the protocol is resistant against such an adversary, and the × indicates that the protocol is vulnerable to an attacker having such a characteristic.

Protocol	C1	C2	C3
ZeroProKeS	✓	✓	✓
QTAB-KEP [5]	×	✓	✓
Diffie Hellman [59]	×	✓	✓
Common Randomness [14]	×	✓	✓
Key Wrapping [60]	✓	✓	×

TABLE 5: Comparison among protocols against adversary characteristics

9.2.2 Comparison of Utility Requirements

The utility requirements ensure the ease of usability of a protocol. The first four requirements (U1 - U4) ensure the minimization of a device effort even in the scenario of a large-scale deployment. The rest two (U5 and U6) ensure that a key is established even if some clues are missed by a device or modified by an adversary. Table 6 summarizes the result where ✓ points out that the protocol meets the utility requirement, and × illustrates that such a protocol does not meet such a requirement.

Protocol	U1	U2	U3	U4	U5	U6
ZeroProKeS	✓	✓	✓	✓	✓	✓
QTAB-KEP [5]	✓	✓	✓	✓	✓	×
Diffie Hellman [59]	✓	×	✓	✓	×	×
Common Randomness [14]	✓	✓	✓	×	✓	×
Key Wrapping [60]	×	✓	×	✓	✓	✓

TABLE 6: Comparison among protocols for utility requirements

9.2.3 Comparison of Evaluation Metrics

Finally, we compare each of the protocols against several evaluation metrics. E1 ensures that the protocol does not need to maintain a list of keys. Both E2 and E3 assure that the protocol completes successfully and can prevent any known attacks. Both E4 and E5 ensure the protocol authenticates the user and is start time-independent, whereas E6 ensures the forward secrecy of the protocol. The details of the analysis of these metrics are summarized in Table 7.

Protocol	E1	E2	E3	E4	E5	E6
ZeroProKeS	✓	✓	✓	✓	✓	✓
QTAB-KEP [5]	✓	×	✓	✓	✓	✓
Diffie Hellman [59]	✓	×	✓	✓	✓	✓
Common Randomness [14]	✓	×	✓	×	✓	✓
Key Wrapping [60]	×	✓	✓	×	✓	✓

TABLE 7: Comparison among protocols for evaluation metrics

9.3 Discussion on the application scenario

It is to be noted that our threat model may not be completely applicable for some body-area network applications. For example, in several body-area network applications, a new device is expected to be integrated into the already established network quickly. These scenarios are not directly related to the extent of the protocol. However, the protocol can still be utilized in such scenarios, albeit with little human input. The new device can be initialized with a "weaker key", and the devices, already interconnected in the same network, will share minimal information with the new device. Once the new device gathers enough clues to generate a "stronger key", it will be added to the network, and the devices can start sharing sensitive information, encrypted by this "stronger key." Such a modification will make our proposed protocol "Quasi-Effortless" (see usability requirement U4).

10 Conclusion

This paper proposes *ZeroProKeS*, an automatic key establishment protocol between the two legitimate parties who enjoy more quality time with each other than the attacker. The protocol applies the set intersection method and erasure coding to achieve such a feat. It identifies the limitation of the basic QTAB-KEP protocol and improves it. Moreover, we have also formalized the definition of robustness, security, and resilience for the protocol and evaluated it against these definitions to provide the limit for all of them. Furthermore,

we have shown how the protocol is impervious to many possible attacks. The paper also shows how to achieve any arbitrary information transfer function by adhering to the proposed protocol. We have demonstrated that the *ZeroProKeS* algorithm can securely pair two devices through the implementation between Raspberry pi and an Android app. We have found that the computational cost of the protocol is comparable to the QTAB-KEP protocol, even though our protocol is also resilient against an active attacker. Moreover, our proposed protocol only draws an additional 2.478% of the benchmark power of Raspberry pi. Finally, we have shown that the computational load for performing the key establishment, using *ZeroProKeS*, decays exponentially with the increase of time for key establishment, making the protocol suitable for low-cost, large-scale applications.

REFERENCES

[1] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov, "Using frankencerts for automated adversarial testing of certificate validation in ssl/tls implementations," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 114–129.

[2] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "You get where you're looking for: The impact of information sources on code security," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 289–305.

[3] D. H. Steinberg and S. Cheshire, *Zero Configuration Networking: The Definitive Guide: The Definitive Guide*. " O'Reilly Media, Inc.", 2005.

[4] G. T. Amariucai, C. Bergman, and Y. Guan, "An automatic, time-based, secure pairing protocol for passive rfid," in *International Workshop on Radio Frequency Identification: Security and Privacy Issues*. Springer, 2011, pp. 108–126.

[5] G. T. Amariucai, S. Barman, and Y. Guan, "An algebraic quality-time-advantage-based key establishment protocol," in *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, 2018, pp. 144–153.

[6] G. McFarlane, *How Toyota Makes Money*, 2019 (accessed March 3, 2021), https://www.investopedia.com/articles/markets/021416/how-toyota-makes-money-tm.asp.

[7] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[8] Y. Li, "Design of a key establishment protocol for smart home energy management system," in *2013 Fifth International Conference on Computational Intelligence, Communication Systems and Networks*. IEEE, 2013, pp. 88–93.

[9] Q. Huang, J. Cukier, H. Kobayashi, B. Liu, and J. Zhang, "Fast authenticated key establishment protocols for self-organizing sensor networks," in *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, 2003, pp. 141–150.

[10] I. Csiszár and J. Korner, "Broadcast channels with confidential messages," *IEEE transactions on information theory*, vol. 24, no. 3, pp. 339–348, 1978.

[11] A. D. Wyner, "The wire-tap channel," *Bell system technical journal*, vol. 54, no. 8, pp. 1355–1387, 1975.

[12] M. R. Khalili-Shoja, G. T. Amariucai, S. Wei, and J. Deng, "Secret common randomness from routing metadata in ad hoc networks," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 8, pp. 1674–1684, 2016.

[13] M. R. K. Shoja, G. T. Amariucai, Z. Wang, S. Wei, and J. Deng, "Asymptotic converse bound for secret key capacity in hidden markov model," in *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 1968–1972.

[14] A. Khisti, S. N. Diggavi, and G. W. Wornell, "Secret-key generation using correlated sources and channels," *IEEE Transactions on Information Theory*, vol. 58, no. 2, pp. 652–670, 2012.

[15] C. Ye, S. Mathur, A. Reznik, Y. Shah, W. Trappe, and N. B. Mandayam, "Information-theoretically secret key generation for fading wireless channels," *IEEE Transactions on Information Forensics and Security*, vol. 5, no. 2, pp. 240–254, 2010.

[16] K. Ren, H. Su, and Q. Wang, "Secret key generation exploiting channel characteristics in wireless communications," *IEEE Wireless Communications*, vol. 18, no. 4, pp. 6–12, 2011.

[17] M. Miettinen, N. Asokan, T. D. Nguyen, A.-R. Sadeghi, and M. Sobhani, "Context-based zero-interaction pairing and key evolution for advanced personal devices," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 880–891.

[18] J. Han, M. Harishankar, X. Wang, A. J. Chung, and P. Tague, "Convoy: Physical context verification for vehicle platoon admission," in *Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications*, 2017, pp. 73–78.

[19] D. Schürmann and S. Sigg, "Secure communication based on ambient audio," *IEEE Transactions on mobile computing*, vol. 12, no. 2, pp. 358–370, 2011.

[20] J. Han, A. J. Chung, M. K. Sinha, M. Harishankar, S. Pan, H. Y. Noh, P. Zhang, and P. Tague, "Do you feel what i hear? enabling autonomous iot device pairing using different sensor types," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 836–852.

[21] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," in *Proceedings of the 9th ACM conference on Computer and communications security*, 2002, pp. 41–47.

[22] W. Du, J. Deng, Y. S. Han, and P. K. Varshney, "A key predistribution scheme for sensor networks using deployment knowledge," *IEEE Transactions on dependable and secure computing*, vol. 3, no. 1, pp. 62–77, 2006.

[23] T. Ito, H. Ohta, N. Matsuda, and T. Yoneda, "A key predistribution scheme for secure sensor networks using probability density function of node deployment," in *Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks*, 2005, pp. 69–75.

[24] A. Juels, "Minimalist cryptography for low-cost rfid tags," in *International conference on security in communication networks*. Springer, 2004, pp. 149–164.

[25] G. Karjoth and P. A. Moskowitz, "Disabling rfid tags with visible confirmation: clipped tags are silenced," in *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, 2005, pp. 27–30.

[26] E. EPCglobal and K. Chiew, "Radio-frequency identity protocols class-1 generation-2 uhf rfid protocol for communications at 860 mhz–960 mhz version 1.0. 9," *K. Chiew et al./On False Authenticationsfor C1G2 Passive RFID Tags*, vol. 65, 2004.

[27] S. Mathur, W. Trappe, N. Mandayam, C. Ye, and A. Reznik, "Radio-telepathy: extracting a secret key from an unauthenticated wireless channel," in *Proceedings of the 14th ACM international conference on Mobile computing and networking*, 2008, pp. 128–139.

[28] U. M. Maurer, "Secret key agreement by public discussion from common information," *IEEE transactions on information theory*, vol. 39, no. 3, pp. 733–742, 1993.

[29] R. Ahlswede and I. Csiszár, "Common randomness in information theory and cryptography. i. secret sharing," *IEEE Transactions on Information Theory*, vol. 39, no. 4, pp. 1121–1132, 1993.

[30] C. Cachin and U. M. Maurer, "Linking information reconciliation and privacy amplification," *journal of Cryptology*, vol. 10, no. 2, pp. 97–110, 1997.

[31] C. Ye, A. Reznik, and Y. Shah, "Extracting secrecy from jointly gaussian random variables," in *2006 IEEE International Symposium on Information Theory*. IEEE, 2006, pp. 2593–2597.

[32] J. Cardinal and G. Van Assche, "Construction of a shared secret key using continuous variables," in *Proceedings 2003 IEEE Information Theory Workshop (Cat. No. 03EX674)*. IEEE, 2003, pp. 135–138.

[33] S. Jarecki, H. Krawczyk, and J. Xu, "Opaque: an asymmetric pake protocol secure against pre-computation attacks," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 456–486.

[34] J. Becerra, D. Ostrev, and M. Škrobot, "Forward secrecy of spake2," in *International Conference on Provable Security*. Springer, 2018, pp. 366–384.

[35] F. Hao, "J-pake: Password-authenticated key exchange by juggling," *RFC 8236*, 2017.

[36] J. Ding, S. Alsayigh, J. Lancrenon, S. RV, and M. Snook, "Provably secure password authenticated key exchange based on rlwe for the post-quantum world," in *Cryptographers' Track at the RSA conference*. Springer, 2017, pp. 183–204.

[37] X. Gao, J. Ding, L. Li, R. Saraswathy, and J. Liu, "Efficient implementation of password-based authenticated key exchange from rlwe and post-quantum tls," *Cryptology ePrint Archive*, 2017.

[38] M. Carbone, V. Conin, M.-A. Cornelie, F. Dassance, G. Dufresne, C. Dumas, E. Prouff, and A. Venelli, "Deep learning to evaluate

secure rsa implementations," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 132–161, 2019.
[39] T. Roche, V. Lomné, C. Mutschler, and L. Imbert, "A side journey to titan," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 231–248.
[40] Q. Xie, D. S. Wong, G. Wang, X. Tan, K. Chen, and L. Fang, "Provably secure dynamic id-based anonymous two-factor authenticated key exchange protocol with extended security model," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 6, pp. 1382–1392, 2017.
[41] G. Yang, D. S. Wong, H. Wang, and X. Deng, "Two-factor mutual authentication based on smart cards and passwords," *Journal of computer and system sciences*, vol. 74, no. 7, pp. 1160–1172, 2008.
[42] Q. Wang, D. Wang, C. Cheng, and D. He, "Quantum2fa: efficient quantum-resistant two-factor authentication scheme for mobile devices," *IEEE Transactions on Dependable and Secure Computing*, 2021.
[43] D. Wang and P. Wang, "Two birds with one stone: Two-factor authentication with security beyond conventional bound," *IEEE transactions on dependable and secure computing*, vol. 15, no. 4, pp. 708–722, 2016.
[44] E. Erdem and M. T. Sandıkkaya, "Otpaas—one time password as a service," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 743–756, 2018.
[45] J. W. Byun, "A generic multifactor authenticated key exchange with physical unclonable function," *Security and Communication Networks*, vol. 2019, 2019.
[46] M. Gupta and N. S. Chaudhari, "Anonymous two factor authentication protocol for roaming service in global mobility network with security beyond traditional limit," *Ad Hoc Networks*, vol. 84, pp. 56–67, 2019.
[47] X. Li, J. Peng, M. S. Obaidat, F. Wu, M. K. Khan, and C. Chen, "A secure three-factor user authentication protocol with forward secrecy for wireless medical sensor network systems," *IEEE Systems Journal*, vol. 14, no. 1, pp. 39–50, 2019.
[48] S. Pirandola, U. L. Andersen, L. Banchi, M. Berta, D. Bunandar, R. Colbeck, D. Englund, T. Gehring, C. Lupo, C. Ottaviani *et al.*, "Advances in quantum cryptography," *Advances in optics and photonics*, vol. 12, no. 4, pp. 1012–1236, 2020.
[49] J. Ding, X. Xie, and X. Lin, "A simple provably secure key exchange scheme based on the learning with errors problem," *Cryptology ePrint Archive*, 2012.
[50] J. Becerra, V. Iovino, D. Ostrev, P. Šala, and M. Škrobot, "Tightly-secure pak (e)," in *International Conference on Cryptology and Network Security*. Springer, 2017, pp. 27–48.
[51] Z. Li and D. Wang, "Two-round pake protocol over lattices without nizk," in *International Conference on Information Security and Cryptology*. Springer, 2018, pp. 138–159.
[52] X. Gao, J. Ding, J. Liu, and L. Li, "Post-quantum secure remote password protocol from rlwe problem," in *International Conference on Information Security and Cryptology*. Springer, 2017, pp. 99–116.
[53] A. S. Fraenkel and Y. Yesha, "Complexity of problems in games, graphs and algebraic equations," *Discrete Applied Mathematics*, vol. 1, no. 1-2, pp. 15–30, 1979.
[54] A. Juels, "Secret sharing in cryptographic devices via controlled release of plaintext information," Jul. 8 2014, uS Patent 8,774,410.
[55] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the society for industrial and applied mathematics*, vol. 8, no. 2, pp. 300–304, 1960.
[56] O. Kotevska, K. Perumalla, and J. Lopez, "Kensor: Coordinated intelligence from co-located sensors," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 4169–4174.
[57] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 553–567.
[58] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the society for industrial and applied mathematics*, vol. 8, no. 2, pp. 300–304, 1960.
[59] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.
[61] A. J. Menezes, J. Katz, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.
[60] P. Rogaway and T. Shrimpton, "Deterministic authenticated-encryption," in *Advances in Cryptology–EUROCRYPT*, vol. 6. Citeseer, 2007.
[62] A. Karatsuba, "Multiplication of multidigit numbers on automata," in *Soviet physics doklady*, vol. 7, 1963, pp. 595–596.
[63] R. Pi, "Raspberry pi 3 model b," *online].(https://www. raspberrypi. org*, 2015.
[64] A. Studio, "Android studio," *The Official IDE for Android*, 2017.
[65] Adafruit, *USB Charger Doctor - In-line Voltage and Current Meter*, 2021 (accessed March 3, 2021), https://www.amazon.com/dp/B00NAY2R2W/ref=tsm/1/fb/lk.
[66] B. Castle, *The Legion of the Bouncy Castle*, 2020 (accessed March 3, 2021), https://www.bouncycastle.org/latest_releases.html.

Shahnewaz Karim Sakib received his B.Sc. degree in electrical engineering from Bangladesh University of Engineering and Technology, in 2015. He is currently pursuing his Ph.D. degree with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA. His research interest are focused on information theoretic privacy, machine learning, and wireless communication networks.

George T Amariucai was born and raised in Romania. He received his PhD in electrical and computer engineering from Louisiana State University (2009).

Amariucai is currently with the Department of Computer Science at Kansas State University. His research interests are focused on cyber security and its intersections with probability and information theory, applied and theoretical machine learning, wireless communication networks, cryptography, and social sciences. He is the director of the PITS Lab.

Yong Guan Dr. Yong Guan is a professor of Electrical and Computer Engineering, the Associate Director for Research of Information Assurance Center at Iowa State University, and Cyber Forensics Coordinator of the NIST Center of Excellence in Forensic Sciences – CSAFE. He received his Ph.D. degree in Computer Science from Texas A&M University in 2002, MS and BS degrees in Computer Science from Peking University in 1996 and 1990, respectively. With the support of NSF, IARPA, NIST, and ARO, his research focuses on security and privacy issues, including digital forensics, network security, and privacy-enhancing technologies for the Internet. The resulted solutions have addressed issues in attack attribution, secure network coding, key management, localization, computer forensics, anonymity, and online frauds detection. He served as the general chair of 2008 IEEE Symposium on Security and Privacy (Oakland 2008, the top conference in security), co-organizer for ARO Workshop on Digital Forensics, and the co-coordinator of Digital Forensics Working Group at NSA/DHS CAE Principals Meetings. Dr. Guan has been recognized by awards including NSF Career Award, ISU Award for Early Achievement in Research, the Litton Industries Professorship, and the Outstanding Community Service Award of IEEE Technical Committee on Security and Privacy.