Toward a Resilient Key Exchange Protocol for IoT

Zhangxiang Hu
University of Oregon
Eugene, Oregon, USA
huz@cs.uoregon.edu

Jun Li
University of Oregon
Eugene, Oregon, USA
lijun@cs.uoregon.edu

Samuel Mergendahl
University of Oregon
Eugene, Oregon, USA
smergend@cs.uoregon.edu

Christopher Wilson
University of Oregon
Eugene, Oregon, USA
cwilson@cs.uoregon.edu

ABSTRACT

In order for resource-constrained Internet of Things (IoT) devices to set up secure communication channels to exchange confidential messages, Symmetric Key Cryptography (SKC) is usually preferred to resource-intensive Public Key Cryptography (PKC). At the core of setting up a secure channel is secure key exchange, the process of two IoT devices securely agreeing on a common session key before they communicate. While compared to using PKC, key exchange using SKC is more resource-aware for IoT environments, it requires either a pre-shared secret or *trusted* intermediaries between the two devices; neither assumption is realistic in IoT.

In this paper, we relax the above assumptions and introduce a new intermediary-based secure key exchange protocol for IoT devices that do not support PKC. With a design that is lightweight and deployable in IoT, our protocol fundamentally departs from existing intermediary-based solutions in that (1) it leverages intermediary parties that can be malicious and (2) it can detect malicious intermediary parties. We provide a formal proof that our protocol is secure and conduct a theoretical analysis to show the failure probability of our protocol is easily negligible with a reasonable setup and its malicious helper detection probability can be 1.0 even when a malicious helper only tampers a small number of messages. We implemented our protocol and our experimental results show that our protocol significantly improves the computation time and energy cost. Dependent on the IoT device type (Raspberry Pi, Arduino Due, or Sam D21) and the PKC algorithms to compare against (ECDH, DH, or RSA), our protocol is 2.3 to 1591 times faster on one of the two devices and 0.7 to 4.67 times faster on the other.

CCS CONCEPTS

• **Security and privacy → Key management**; **Security protocols**; • **Networks** → *Security protocols*; • **Theory of computation → Cryptographic protocols**.

CODASPY '22, April 24–27, 2022, Baltimore, MD, USA

ACM ISBN 978-1-4503-9220-4/22/04…$15.00
https://doi.org/10.1145/3508398.3511520

KEYWORDS

Internet of things (IoT), key exchange, helper party, malicious intermediary

ACM Reference Format:
Zhangxiang Hu, Jun Li, Samuel Mergendahl, and Christopher Wilson. 2022. Toward a Resilient Key Exchange Protocol for IoT. In *Proceedings of the Twelveth ACM Conference on Data and Application Security and Privacy (CODASPY '22), April 24–27, 2022, Baltimore, MD, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3508398.3511520

1 INTRODUCTION

Due to advances in lightweight computing and networking technologies, the Internet of Things (IoT) has rapidly penetrated into our lives. However, because a compromised IoT system can lead to disastrous results [16, 26, 30], a key challenge facing IoT is that IoT networks must support secure communications channels to protect message integrity and confidentiality, thus resistant to both message tampering and eavesdropping. While IoT devices can either employ public key cryptography (PKC) or symmetric key cryptography (SKC) to establish secure communication channels between them, due to their often extremely constrained resources and computing power, many IoT devices are not capable of performing PKC and have to resort to SKC. A central question with using SKC, however, is key exchange; that is, any two IoT devices must exchange a common secret key in order to encrypt and decrypt messages between them.

Non-cryptographic solutions have been proposed for secret key exchange between IoT devices. A typical solution is using a secure secondary communication channel, which however usually requires additional hardwares or sensors [17, 29] that IoT devices may not be equipped with. Other non-cryptographic solutions include jamming [2] and proximity [22]. The jamming solution requires a special entity—*jammer*—to jam the channel and the proximity solution needs IoT devices to be physically close to each other (e.g., 6cm); both are often unrealistic.

Cryptographic key exchange solutions can be various methods using PKC (e.g., Diffie-Hellman, ECC, RSA) or methods not using PKC. The former's demand on resources and computing power is often beyond the reach of IoT devices. The latter are methods using SKC. In contrast to using PKC, SKC-based key exchange has a better performance with significantly lower usage of resources and computational power. There are two approaches in using SKC for key exchange between two parties: using a pre-shared secret between the two parties, or using the help of intermediary helper parties

between the two parties. As an IoT network is often composed of hundreds or even thousands of devices, doing the former approach for every pair of devices is daunting. The latter approach is more feasible, which we focus on in this paper.

All existing intermediary-based key exchange protocols must trust the intermediaries, a stringent and often unrealistic requirement. If the intermediary helper parties are compromised and tamper messages from the key exchange parties, IoT devices may not detect the compromise and they may either fail to exchange a secret key between them or leak useful information pertaining to the key to adversaries. Key exchange parties could try to sign their messages, but signing with PKC is too expensive for IoT devices, and signing with SKC requires the key exchange parties to have a shared key between them which they have yet to agree on.

In this work, we design, prove, and evaluate a new intermediary-based key exchange protocol for devices with limited resources—especially IoT devices—to successfully and securely agree upon a secret session key. In particular, we apply the cut-and-choose technique to identify the malicious helpers without using any PKC primitives. Cut-and-choose is widely adopted in multi-party computation (MPC) [1, 18] to achieve security against malicious parties. Its main idea is to let one party construct different versions of a secret message and have the other party randomly check some of them and use the rest of them. In our protocol, we first let an IoT device create a bunch of test keys, and then let the other IoT device randomly pick a subset of test keys to detect malicious helpers and use the remaining test keys to derive a real secret session key for communication between the two devices. Our main contributions include:

- Our protocol advances SKC-based key exchange. Unlike any previous intermediary-based solution, our protocol is the first one that does *not* rely on the trustworthiness of helper parties. Also, the protocol does not leak any useful information to the helper parties. If some helpers are malicious and do not follow the protocol, the two devices will still be able to establish a session key without leaking any useful information.
- Our protocol introduces a novel design that can efficiently identify the malicious helpers when they tamper messages going through them, even if they collude or selectively tamper messages.
- With the SK-security framework, we formally prove that our protocol is secure against malicious intermediary helpers.
- We conduct theoretical analysis of our protocol and show its failure probability is easily negligible with a reasonable setup and its malicious helper detection probability can be 1.0 even when a malicious helper only tampers a small number of messages.
- We provide empirical evaluations for our protocol. We implemented our protocol and emulated different IoT devices on Mininet to evaluate its performance against three widely used PKC-based protocols: RSA, Diffie-Hellman, and Elliptic Curve Diffie-Hellman. For two parties doing key exchange, our experiments demonstrated that our protocol achieves 2.3 to 1591 times faster on one party and 0.7 to 4.67 times faster on the other.

The rest of this paper is organized as follows. Section 2 reviews the related work on previous key exchange protocols in IoT environments. Section 3 introduces the basic design of the intermediary-based key exchange protocol with secret sharing scheme. Section 4 describes our new mechanism to detect malicious behaviors and identify cheating intermediaries. Section 5 proves the security of our protocol against malicious intermediaries. Section 6 provides a theoretical analysis of our protocol's efficacy, resiliency, and overhead. Section 7 shows the experimental results of our protocol's performance and network overhead. Finally, Section 8 concludes this paper.

2 RELATED WORK

A secure key exchange protocol is a core cryptographic primitive in building secure communication channels [7]. Various standard public key cryptography (PKC) schemes are sufficient to implement a secure key exchange protocol in traditional networks. However, due to the limited resources of IoT devices, these schemes are not suitable for many IoT environments. Many previous approaches were introduced to improve the efficiency of PKC, such as more efficient variants of Elliptic Curve schemes [6, 9]. The computational cost *during* key exchange can also be reduced by performing pre-computations *before* key exchange [21]. However, improvements on PKC-based methods are limited, mostly insufficient in addressing the resource limitations of IoT devices. Below we focus on previous approaches that mainly use SKC.

One key exchange solution without PKC is using a pre-shared secret. For example, the approach in [15] and [27] assume that all nodes in the same network share a common master key, from which any two nodes can derive their session key. However, if any node is compromised, it will expose the master key and therefore threaten the confidentiality of the entire network. To address this issue, some approaches (such as those in [10, 20]) instead use a password between a client and all its servers as a pre-shared secret, where every server has a share of the password. The servers collectively use the password to authenticate the client and then derive a session key for the client to communicate with any one of the servers. Here, unless more than a threshold number of servers are compromised, a compromised server node will not leak the password. Unfortunately, these password-based approaches still employs PKC. Also, like the pre-shared master key, they still have a single point-of-failure (the password), and they cannot identify which server(s), if any, are compromised.

Instead of one common pre-shared secret among all nodes, Chan *et al.* [12] suggest each node pre-store a set of keys randomly selected from a universal key space, where the sets of any two nodes overlap. When a node decides to start a communication session with another node, it must identify all the common keys it shares with that node and then derive a session key between them from the common keys. If an attacker subverts a node, the attacker can only learn the keys in the node's set of keys, while the session key remains secure. However, the procedure to identify common keys between different nodes could leak useful information about the universal key space and eventually the information of the session keys between nodes. In a similar work [19], every node is associated with a set of polynomials in a universal pool of random bivariate

polynomials. Any two nodes need to derive their session key by first identifying their common bivariate polynomials, which however could leak useful information of the pool and also the information of the session keys.

Different from using a pre-shared secret, another solution is to use help from a trusted third party. Hummen *et al.* [13] suggested that as long as an IoT device maintains a key associated with an external trusted server, it then can use the help of the trusted server to derive a new secret session key for its communication with another party. This approach drastically reduces the computations of IoT devices. Yet, the trusted server is a major point of failure. If it is compromised, it could obtain all secret keys.

Instead of placing trust into a single third party, researchers proposed solutions using multiple intermediary helpers. Solutions in [14, 23–25] use the neighboring nodes of key exchange parties as intermediary helpers, whereas for the solution in [12], multiple independent communication paths between two communication nodes can be regarded as intermediary helpers. A party can initiate a key exchange with another party by splitting a *secret* into multiple *secret shares* and sending each share to a different intermediary helper, where each share leaks no information of the original secret. Every intermediary helper then forwards the share it receives to the other key exchange party, which subsequently assembles all the shares it receives to derive the original secret, and both parties can then use the same secret to derive their session key. However, these intermediary-based solutions assume all intermediaries are trusted or at least *semi-honest*. In other words, *all* intermediaries must follow the protocol honestly. If any intermediary becomes malicious and deviates from the protocol, such as discarding a secret share or tampering a secret share before forwarding it, the whole key exchange could fail and the malicious intermediary may learn certain information of the secret, potentially weakening the confidentiality strength of the session key. Furthermore, the communication nodes cannot detect which intermediary helpers are compromised by the adversary.

3 BASIC DESIGN

Not only does our intermediary-based key exchange solution eliminate all PKC operations and only rely on SKC operations, it also significantly differs from prior intermediary-based solutions and adds new features. In particular, we describe the basic design of our key exchange solution in this section and focus on the resiliency against malicious intermediaries in the next section.

3.1 Settings and Assumptions

Every IoT device, say P_A, communicates with another IoT device, say P_B, via a public channel, which is not secure as messages through the channel could be eavesdropped or tampered. P_A and P_B thus need to exchange a session key to protect their communication, where P_A is the **key exchange initiator** and P_B is **key exchange responder**. P_A and P_B are honest and follow their key exchange protocol between themselves. Finally, both parties are resource-constrained IoT devices and can only perform SKC operations (i.e., no PKC operations).

Between P_A and P_B are n intermediary helper parties H_i ($i = 1, \ldots, n$) (Figure 1) that will assist the key exchange. A helper can be a gateway device, a smart phone, or another IoT device. Further, P_A and P_B each set up a secure channel with every helper through a registration process, which can establish a shared secret between an IoT device and a helper and use the shared secret to set up a secure channel between them for their communication. (Note this registration process is not suitable for two IoT devices to exchange a session key as it will need to register every IoT device at its every communication party, a much larger overhead than registering a device at all its helpers.) Finally, unlike P_A and P_B who are honest, a helper may be malicious. We assume there are less than t helpers in total which are malicious.

Before they start key exchange, P_A and P_B authenticate each other, as follows. For P_A to authenticate itself to P_B, P_A composes an authentication message about its identity and sends it to every helper (through its secure channel with the helper). Every helper then verifies the message; if the message is verified, the helper then sends a claim to P_B (through its secure channel with P_B) that the other side is indeed P_A. On the side of P_B, upon the receipt of claims from all the helpers, P_B can then decide if P_A is authenticated based on its authentication policy, which, for example, may require (a) all the claims vouch for P_A, or (b) the majority of claims vouch for P_A, or (c) no more than a threshold number or percentage of claims vouch for an identify that is not P_A. Clearly, except for policy (a), if some helpers are malicious, P_B can still authenticate P_A. P_B can authenticate itself to P_A in the same way.

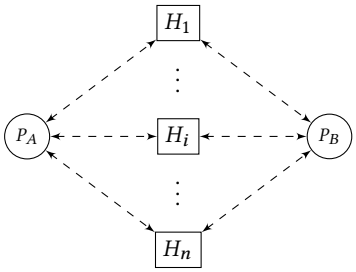


Figure 1: The settings of key exchange. P_A and P_B are communication devices and H_i ($i = 1, \ldots, n$) are intermediary helpers.

3.2 Key Exchange Protocol π

We now describe the key exchange protocol π to illustrate the basic design of our key exchange solution. It leverages a standard *t-out-of-n secret sharing scheme* [28] in which a secret S is composed of n shares and a collection of at least $t (t \leq n)$ shares must be present in order to reconstruct S. Any collection that has less than t shares does not leak any information about S. The main idea of π is for the key exchange initiator P_A to split a secret into n shares and for the key exchange responder P_B to receive at least t shares separately through t helpers and reconstruct the original secret, thus P_A and P_B are able to use the same secret to derive their session key. The protocols is as follows.

(a) **Initialization.** P_A initializes the key exchange with P_B by sending P_B a message (INIT, *sid*) (via a public channel) where INIT contains P_A's security parameters (including ciphers and parameters available for key exchange and ciphers and key lengths for

its communication with P_B) and *sid* is the ID of the current key exchange session. P_B then sends back (INITCONFIRM, *sid*) (via a public channel) where INITCONFIRM contains a subset of P_A's security parameters that P_B agrees with for their key exchange.

(b) **Choose secret and its shares.** P_A randomly choose a secret S and invokes a t-out-of-n secret sharing scheme to obtain n shares of S: $\{s_i | i = 1, \ldots, n\}$.

(c) **Transfer secret shares.** P_A sends s_i to H_i ($i = 1, \ldots, n$), which then forwards s_i to P_B after receiving s_i.

(d) **Derive secret from shares.** Upon receipt t shares among $\{s_i | i = 1, \ldots, n\}$, P_B then uses the t-out-of-n secret sharing scheme to reconstruct S.

(e) **Derive session key.** P_A and P_B both compute $k_{sid} = f(S, 0)$, where f is a pseudorandom function agreed by P_A and P_B during initialization. k_{sid} is then the session key for P_A and P_B.

(f) **Verify session key.** Furthermore, P_A and P_B each compute $S' = f(S, 1)$, and P_B sends an acknowledgement message $M = g(\text{"CONFIRM"}, sid, P_A, P_B, S')$ to P_A where g is a message authentication function (also agreed by P_A and P_B during initialization). Upon the receipt of M, P_A checks if M is also $g(\text{"CONFIRM"}, sid, P_A, P_B, S')$. If so, P_A knows both parties agree on k_{sid} as their session key, and P_A can start its communication with P_B; otherwise, P_A either aborts the protocol or initiates another instance of π.

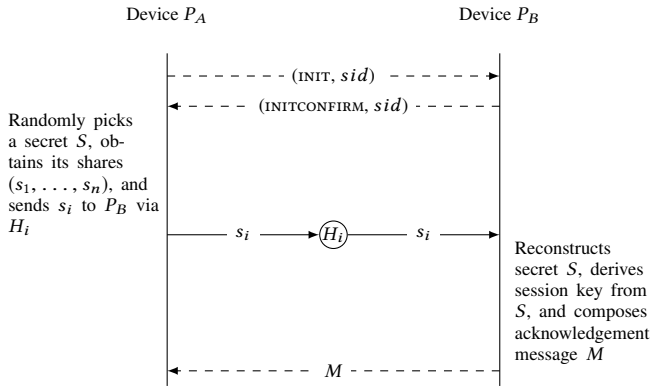


Figure 2: Key exchange protocol π. Each dashed line means a message is sent via a public channel. Each solid line means a message is sent via an intermediary helper party.

4 RESILIENCY DESIGN

4.1 Overview

Protocol π is not resilient against malicious helpers. If a helper tampers or forges a share before sending it to P_B and P_B uses it with other shares to reconstruct the secret (S), P_B will not derive the same secret that P_A has, resulting in the failure of the key exchange. Moreover, P_A and P_B cannot detect or identify malicious helpers. A typical approach to this problem is to sign every share, but signing with PKC is too expensive for IoT devices, and signing with SKC requires P_A and P_B to have a session key between them already, which they have yet to agree on.

We design a new protocol π^A that advances π with resiliency. Without using any PKC operation, π^A enables key exchange devices to try to detect and identify malicious helpers. The main design idea of π^A is derived from the cut-and-choose technique widely used in secure multi-party computation. The cut-and-choose technique lets one party construct different versions of a message and have the other party randomly checks some of them and use the rest of them. In π^A, P_A generates a number of random keys which we call **test keys**, P_B use some of them called **opening keys** to identify malicious helpers via an efficient and effective design, and P_A and P_B use the rest of them called **evaluation keys** to derive the session key.

4.2 Key Exchange Protocol π^A: General Design

π^A is composed of three phases. We overview them here and elaborate them in Section 4.3.

Initialization phase. As opposed to choosing one secret S as in π, P_A now generate a number of test keys. For every test key, π^A invokes a standard t-out-of-n secret sharing scheme to split it into n shares, sends each share to a different helper, which then forwards the share to P_B. Note that with the assumption that there are less than t helpers in total which are malicious (Section 3.1), the security property of the t-out-of-n secret sharing scheme guarantees that the malicious helpers, even if they collude, will not be able to have t or more shares to learn any useful information of any test key.

Cut-and-choose phase. This phase is focused on identifying malicious helpers and drops shares from them. P_B first randomly chooses half of the test keys as opening keys and the other half test keys as evaluation keys and also notifies P_A its choice. P_A then retransmits a copy of every share of every opening key to P_B via a helper rather than the original helper that forwarded the share, where the helper is randomly chosen each time. P_B then inspects every helper and compares every share of an opening key forwarded by the helper against the share's copy retransmitted via another helper. If there are t or more helpers that disagree with the helper, P_B then regards the helper as malicious. Otherwise, i.e., if this helper was *not* malicious, every helper who disagreed with the helper is then malicious; with t or more disagreements, there would be then t or more malicious helpers, which contradicts with the assumption that at most $t - 1$ helpers are malicious (Section 3.1).

If more than n-t helpers are malicious, P_B aborts the protocol. Otherwise, P_B drops all the shares forwarded by every helper identified as malicious, some of which could be shares of an evaluation key. P_B finally reconstructs every evaluation key with its remaining shares. Although it is still likely that some remaining shares are compromised and as a result evaluation keys reconstructed with them are also compromised, the likelihood is low given that most remaining shares are authentic.

Session key derivation phase. P_A randomly chooses a secret, uses each evaluation key to encrypt the secret separately, and sends each encrypted secret to P_B. P_B then uses the corresponding evaluation key to decrypt every encrypted secret. Although P_B may not reconstruct some evaluation keys correctly due to compromised shares, it can treat the decryption output with the majority agreement as the secret. P_A and P_B can therefore use the secret to derive their session key.

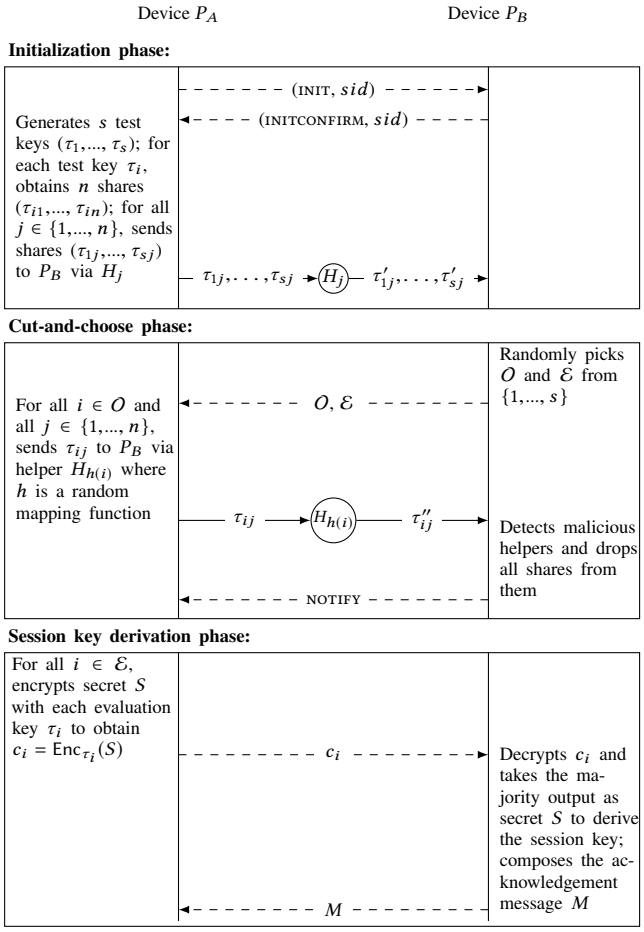


Figure 3: Key exchange protocol π^A. Each dashed line means a message is sent via a public channel. Each solid line means a message is sent via an intermediary helper party.

4.3 Key Exchange Protocol π^A: Protocol

The protocol π^A is as follows.

[Initialization phase.] This phase is the same as π's Initialization (see Section 3.2), except that the INIT also contains the number of test keys from P_A. Plus, P_A sends test keys to P_B as follows:

- P_A randomly generates s test keys $\mathcal{T} = (\tau_1, \tau_2, \cdots, \tau_s)$, where every test key is of an equal length.
- For every $\tau_i \in \mathcal{T}$, P_A invokes the t-out-of-n secret sharing scheme to obtain its n shares $(\tau_{i1}, \tau_{i2}, \cdots, \tau_{in})$.
- For every test key τ_i and its every share τ_{ij}, P_A sends τ_{ij} to helper H_j, which then forwards the share to P_B. Helper H_j will thus receive and forward a set of shares $(\tau_{1j}, \tau_{2j}, \cdots, \tau_{sj})$.
- For each τ_i, P_B receives shares $(\tau'_{i1}, \tau'_{i2}, \cdots, \tau'_{in})$. (We use notation τ'_{ij} instead of τ_{ij} since a share may be tampered by a corrupted helper.)

[Cut-and-choose phase.] P_B now processes all the test key shares it has received:

- Based on the total number of test keys, P_B randomly chooses half of test key indexes, denoted as O, to be the indexes of opening keys and the other half, denoted as $\mathcal{E}$, to be the indexes of evaluation keys. P_B sends $(O, \mathcal{E})$ to P_A (via a public channel).
- On P_A, upon the receipt of O and $\mathcal{E}$, for every τ_{ij} ($i \in O$) it forwarded, retransmit a copy of τ_{ij} to P_B via helper $H_{h(i)}$, where h is a random mapping function and $\forall i \in O, h(i) \neq j$.
- On P_B, for every helper H_j ($j = 1, \ldots, n$), compare every τ'_{ij} ($i \in O$) it received from H_j with its retransmitted copy from helper $H_{h(i)}$ to see if they match. If for helper H_j there are t or more helpers that disagree with H_j, H_j is then a malicious helper and P_B drops all the test key shares from H_j.
- If more than n-t helpers cheated, P_B aborts the protocol. Otherwise, for every $i \in \mathcal{E}$, P_B knows at least t shares from $(\tau'_{i1}, \tau'_{i2}, \cdots, \tau'_{in})$ still remain. With these remaining shares, P_B thus uses the t-out-of-n secret sharing scheme to reconstruct τ'_i. Here, P_B regards τ'_i as τ_i (which may not be the same if at least one share used is tampered but not found in the previous step).
- P_B sends (NOTIFY) to P_A to let P_A enter the next phase (via a public channel).

[Session key derivation phase.] P_A and P_B now generate their session key as follows:

- P_A randomly chooses a secret S, encrypts S with each evaluation key τ_i separately, $i \in \mathcal{E}$, to obtain ciphertext $c_i = \mathsf{Enc}_{\tau_i}(S)$, and sends each c_i to P_B (via a public channel).
- For each ciphertext c_i ($i \in \mathcal{E}$) received, P_B decrypts it using the evaluation key τ'_i.
- P_B takes the majority output from the previous step as the secret S.
- P_A and P_B follow exactly π's "Derive session key" and "Verify session key" steps (see Section 3.2). P_B also notifies P_A the identities of malicious helpers, encrypted with their newly derived session key.

5 SECURITY PROOF OF π^A

We now formally prove the security of protocol π^A. We first introduce the formal definitions of session key security (SK-security) and t-out-of-n secret sharing scheme, and then prove π^A's security.

5.1 Definitions

5.1.1 Session Key Security. We adopt the **session key security (SK-security)** [5], which formally defines the security of a key exchange protocol. We choose this definition because it is conceptually simple and easy to use when analyzing and proving the security of a key exchange protocol. The intuition behind the SK-security is that it means an adversary cannot distinguish a session key from a randomly chosen value.

To define SK-security, we first define a game $\textsc{Game}^{\mathcal{I}}_{\mathcal{A}}$ between a *simulator* $\mathcal{I}$ and an adversary $\mathcal{A}$. Let k be a session key and $c \in \{0, 1\}$ be a coin, $\textsc{Game}^{\mathcal{I}}_{\mathcal{A}}$ is defined in two steps:

- $\mathcal{I}$ first generates the session key k and then tosses the random coin c. $\mathcal{I}$ receives $c \xleftarrow{R} \{0, 1\}$ where $\xleftarrow{R}$ means randomly choosing a value from a set. If c is 0, $\mathcal{I}$ provides the real

session key k to $\mathcal{A}$; otherwise $\mathcal{I}$ randomly chooses a value $k' \xleftarrow{R} \{0,1\}^{|k|}$ from the session key space and returns k' to $\mathcal{A}$.
- With the received value k or k', $\mathcal{A}$ outputs a result c' as its guess for the value c. If $c' = c$ then $\mathcal{I}$ outputs 1 ($\mathcal{I} \rightarrow 1$); otherwise, $\mathcal{I}$ outputs 0 ($\mathcal{I} \rightarrow 0$).

Definition 1. *A key exchange protocol* Π *is SK-secure against adversary* $\mathcal{A}$ *if it satisfies the following properties:*

- *Correctness.* After running Π, the two honest parties establish the same session key only with a negligible probability of failure.
- *Indistinguishability.* The probability that adversary $\mathcal{A}$ outputs a correct c' that equals to c is $\frac{1}{2} + \epsilon(\lambda)$ where $\epsilon(\lambda)$ is a negligible function in λ. Or, in an equivalent expression, assuming $\text{ADV}^{\Pi}_{\mathcal{A}}(\lambda)$ be the advantage of adversary $\mathcal{A}$ to win the game $\text{Game}^{\mathcal{I}}_{\mathcal{A}}$, we then have $\text{ADV}^{\Pi}_{\mathcal{A}}(\lambda) = |Pr[\mathcal{I} \rightarrow 1] - \frac{1}{2}| = \epsilon(\lambda)$.

5.1.2 Secret Sharing Scheme.

Definition 2. *A* t*-out-of-*n *secret sharing scheme* Σ *consists of the following two algorithms:*

- *Share distribution algorithm* Share. A randomized algorithm that takes a secret message m as input and outputs a sequence of n shares: $\mathbb{M} = (m_1, \cdots, m_n)$.
- *Secret reconstruction algorithm* Reconstruct. A deterministic algorithm that takes an input of a collection of t or more shares and outputs the secret message m.

A secure secret sharing scheme should satisfy the property of *correctness* such that for all $U \subseteq \{1, \cdots, n\}$ with $|U| \geq t$, it holds that $Pr[\text{Reconstruct}(m_i | i \in U) = m] = 1$. For any $U \subseteq \{1, \cdots, n\}$ with $|U| < t$, no information will be learned from those shares.

To formalize the security of Σ, let $m, m' \in \mathcal{M}$ be two different messages from the message space $\mathcal{M}$. The challenger (i.e., the simulator) $\mathcal{I}$ invokes the Share algorithm on m, m' and obtains $\mathbb{M} \leftarrow \text{Share}(m)$, $\mathbb{M}' \leftarrow \text{Share}(m')$.

$\mathcal{I}$ also tosses a random coin $b \in \{0,1\}$. If $b = 0$, $\mathcal{I}$ returns $(m_i | i \in U)$ to the adversary $\mathcal{A}$. Otherwise $\mathcal{I}$ returns $(m'_i | i \in U)$. With the received set of shares, $\mathcal{A}$ outputs a result b' as its guess for the value b. If $b' = b$ then $\mathcal{I}$ outputs 1; otherwise, $\mathcal{I}$ outputs 0.

We define the advantage of the adversary $\mathcal{A}$ in this game as:

$$\text{ADV}^{\Sigma}_{\mathcal{A}} = |Pr[\mathcal{I} \rightarrow 1] - \frac{1}{2}|$$

Definition 3. *A* t*-out-of-*n *secret sharing scheme* Σ *is secure over message space* $\mathcal{M}$ *if* $\text{ADV}^{\Sigma}_{\mathcal{A}}$ *is a negligible function.*

An instance of implementation of a t-out-of-n secret sharing scheme is Shamir's secret sharing scheme [28]. The idea behind this scheme is that $d + 1$ points can determine a unique degree-d polynomial. We refer to [28] for more details.

5.2 Security Proof

With SK-security, we first prove that π (specified in Section 3.2) is secure against malicious helpers, and then prove π^A (specified in Section 4.3) is also secure according to an advanced theorem in SK-security.

Proof. We first prove π is secure. We assume in π all helper parties are semi-honest and they follow the protocol and forward messages correctly (i.e., thus messages are authentic). According to Definition 1, to prove this theorem we need to prove both the correctness and the indistinguishability of π.

The correctness of π follows the correctness of the t-out-of-n secret sharing scheme. Since for every $i \in \{1, \ldots, n\}$, helper H_i follows the protocol and forwards s_i correctly, both P_A and P_B will agree on the same secret S. This is guaranteed by the correctness property of a secret sharing scheme defined in Section 5.1.2. It is clear that as P_A and P_B are honest (Section 3.1), they can derive the session key $k_{sid} = f(S, 0)$ with probability one.

To show the indistinguishability property of π, we need to prove *no* adversary has a non-negligible advantage to distinguish a real session key k (i.e., k_{sid} in π) from a random value k'. To do so, we now prove the opposite is not possible. Specifically, we assume that there *was* such an adversary $\mathcal{A}$ against π and show with this assumption, we can construct a distinguisher $\mathcal{D}$ as follows that would violate Definition 3 about the security of the t-out-of-n secret sharing scheme. In another words, $\mathcal{D}$ can distinguish $(s_i | i \in U)$ from $(s'_i | i \in U)$ and output the correct b' with non-negligible probability.

The distinguisher $\mathcal{D}$ works as follows. Upon the input $[k^*, (s_i | i \in U)]$, where k^* is randomly chosen with probability $\frac{1}{2}$ between the real session key k (i.e., k_{sid} in π) and k' (a random string of length k), $\mathcal{D}$ invokes $\mathcal{A}$ which plays the same role as a helper in protocol π. After receiving the share s_i from P_A, $\mathcal{A}$ forwards it to P_B. Based on the input k^*, $\mathcal{A}$ determines whether k^*==k or $k^* \neq k$ and output $c' = 0$ or $c' = 1$, respectively. $\mathcal{D}$ then uses the output of c' from $\mathcal{A}$ as its guess for coin toss b, outputs b, and terminates.

Now we show the contradiction caused by the assumption above. Assume the adversary compromises a helper party and obtains one share from the helper, i.e., $(s_i | i \in U)$. Note that since we assume P_A an P_B are always honest *and* an adversary can only compromise up to $t-1$ helpers, the adversary cannot obtain t shares of the secret. If the real session key k is chosen as the input k^* (i.e., $k^* == k$), s_i is a share of k^*. Otherwise, a random k' is chosen to be k^* and s_i is not a share of k^*. Now, even though k^* is randomly chosen between k and k' with the same probability, $\mathcal{A}$ can guess if the input k^* is the real session key and output the correct c' with non-negligible advantage $\text{ADV}^{\Pi}_{\mathcal{A}}$, therefore $\mathcal{D}$ can base on c' from $\mathcal{A}$ to guess if m_i is a share of k^*, with non-negligible advantage $\text{ADV}^{\Pi}_{\mathcal{A}}$. Clearly, $\mathcal{D}$'s non-negligible advantage contradicts Definition 3. We thus prove the indistinguishability property of π.

Now that we proved both the correctness and the indistinguishability of π, according to Definition 1, π is secure.

Next we prove the security of π^A. We use the theorem that if a key exchange protocol (say Π) in which all key exchange messages are authentic satisfies SK-security, when the protocol is extended to become a new protocol (say Π') in which key exchange messages can be corrupted, the new protocol also satisfies SK-security if it can authenticate messages and discard corrupted ones [5, 8]. Here, when we extend π to π^A, we see in π every message is assumed authentic, while in π^A messages can be tampered by malicious helpers but P_B can identify and drop tampered messages (Section 4.2). Therefore, π^A also satisfies SK-security.

□

6 THEORETICAL PERFORMANCE ANALYSIS OF π^A

In this section we conduct a theoretical performance analysis of π^A. We analyze its failure probability, p_f, the probability that a malicious helper can be detected, p_d, and the number of messages to send during a key exchange session, N.

6.1 Failure Probability (p_f)

π^A fails if P_A and P_B do not reach an agreement on their session key. Note that the failure is only a denial-of-service, while no secret or any useful information is leaked. π^A fails in two cases:

- Case 1: π^A fails if more than n-t helpers are malicious. As described in Sections 4.2 and 4.3, in this case P_B will *not* have enough shares to reconstruct evaluation keys, so it will abort the protocol with $p_f = 1$.
- Case 2: π^A fails if the majority of evaluation keys at P_B are corrupted (i.e., each of them is reconstructed using at least one corrupted share). Denote C the set of corrupted evaluation keys; given there are s test keys and half of them are evaluation keys, we can see in this case $|C| \geq \lceil s/4 \rceil$. As a result, in the session key derivation phase P_B will not be able to correctly decrypt the encrypted secret from P_A and derive the session key.

More specifically, Case 2 happens if $\forall \tau_i \in C$, τ_i would not be selected as an opening key during the cut-and-choose phase, which has a probability of 0.5, and τ_i is not correctly reconstructed. Denote p_r the probability that P_B correctly reconstructs an evaluation key. Now we have:

$$p_f = (0.5 \cdot (1 - p_r))^{|C|} \tag{1}$$

Since $|C| \geq \lceil s/4 \rceil$, we have

$$p_f \leq (0.5 \cdot (1 - p_r))^{\lceil s/4 \rceil} \tag{2}$$

From Equation (2), a higher p_r will result in a lower p_f. Moreover, $0.5 \cdot (1 - p_r)$ is less than 0.5 since p_r is no more than 1. Thus, the failure probability p_f declines exponentially as the number of test keys s increases, which we say p_f is negligible in s.

We now analyze p_r. Let p_c be the cheating probability of each one of the n helpers. The expected number of cheating parties is then $n \cdot p_c$. For each test key, P_B receives $n - (n \cdot p_c)$ correct shares. To reconstruct a test key, P_B needs to choose t correct shares. We thus have:

$$p_r = \prod_{i=0}^{t-1} \frac{n - i - n \cdot p_c}{n - i} \tag{3}$$

Note that for simplicity, here we assume all helpers have the same cheating probability p_c. If each helper H_i has a different cheating probability p_c^i, the expected number of cheating helpers is $\sum_{i=1}^{n} p_c^i$ rather than $n \cdot p_c$.

From Equation (3), p_r is affected by n, t, and p_c. If t and p_c are fixed, when n increases, p_r also increases. This is consistent with the intuition that if there are fixed number of malicious shares, increasing n means more helpers and thus more shares per evaluation key, which provides P_B a better chance to pick correct shares to reconstruct evaluation keys. On the other hand, if n and p_c are fixed, when t increases, p_r would decrease. This is because increasing t requires P_B to select extra shares to reconstruct every evaluation key, which means P_B would have a higher likelihood to pick malicious shares. Finally, if fixing n and t, a higher p_c would cause P_B to have a higher probability to pick malicious shares, thus decreasing p_r.

Finally, combines Equations (2) and (3), if p_f must be lower than an upper bound, while key exchange parties probably cannot control the value of p_c, they can adjust the values of parameters s, t, and n to meet the requirement.

6.2 Malicious Helper Detection Probability (p_d)

Now we discuss the probability that P_B can identify a malicious helper. We point out that if the number of test keys s and the t parameter in π^A's t-out-of-n secret sharing scheme satisfy that $s \geq 4t - 4$, P_B can always identify a malicious helper if it tampered at least $2t - 2$ shares in total of all opening keys We detail the analysis below.

In the cut-and-choose phase, for every helper H_j ($j = 1, ..., n$) P_B counts the number of other helpers that disagrees with the helper in forwarding an opening key's share and identifies the helper as malicious if there are at least t helpers that disagrees with H_j. Below we analyze the probability p_d that P_B can successfully identify a malicious helper H_j based on the number of shares that H_j tampered, Z. Recall every helper forwards one share per opening key, thus forwarding totally $s/2$ shares; clearly, $Z \leq s/2$.

(1) *H_j tampered at least $2t - 2$ shares of opening keys* (i.e., $Z \geq 2t - 2$). Here, because for each share tampered by H_j, P_A retransmitted a copy of its original value along a different helper, i.e., totally at least $2t - 2$ helpers, even if all malicious helpers collude with H_j to not show disagreements (i.e., retransmitting a copy of a share's tampered value rather than its original value), given there are at most $t - 1$ malicious helpers (including H_j), there are at least t benign helpers each of which will disagree with H_j, thus identifying H_j as malicious. i.e., $p_d = 1$.
Notice this case assumes $Z \geq 2t - 2$. Given $Z \leq s/2$, we can obtain that s and t must satisfy $s \geq 4t - 4$.

(2) *H_j tampered less than t shares of opening keys* (i.e., $Z < t$). In this case, P_B cannot identify H_j as malicious. I.e., $p_d = 0$. This is because H_j could be either benign or malicious. Specifically, while it is possible that H_j is malicious and all helpers that disagree with H_j are either benign or malicious, it is also possible that H_j is benign and all helpers that disagree with H_j, whose total number is less than t, are malicious. On the other hand, even though P_B cannot identify H_j as malicious in this case, the number of opening key shares that H_j can tamper must be less than t. Given P_B's random choice of opening keys and evaluation keys from the test keys, the number of evaluation key shares that H_j can tamper must also be less than t on average. Compared to totally $s/2$ shares of all $s/2$ evaluation keys (one share per key) that H_j could have tampered, t is much less than $s/2$ as we set $s \geq 4t - 4$ from (1) above. P_B would thus have a much higher probability to reconstruct evaluations keys correctly, thereby reducing the failure probability p_f.

(3) *H_j tampered $t \leq Z \leq 2t - 3$ shares of opening keys.* In this case, P_B can identify a malicious helper with probability

p_d and we show how to compute p_d as follows. Given that there are $s/2$ opening keys and H_j forwarded the j-th share of every opening key, H_j forwarded totally $s/2$ shares. As P_A retransmitted each of these shares via a randomly chosen helper that is not H_j, we assume the total number of such helpers is Q. Clearly, $Q \leq s/2$. P_B will then check if each of these Q helpers disagrees with H_j, and determines H_j to be malicious if there are at least t disagreements. Denote x the number of disagreements. Assume the worst case where there are $t-1$ malicious helpers and they collude, while there are $n-t+1$ benign helpers (with totally n helpers) and $n-t+1 > t-1$ (or $n-t+1 \geq t$). To detect H_j is malicious, all x disagreements then must come from benign helpers, which has a probability

$$\frac{\binom{n-t+1}{x} \cdot \binom{t-2}{Q-x}}{\binom{n-1}{Q}}.$$

Here, while all Q helpers come from totally $n-1$ helpers (excluding H_j), x helpers are chosen from $n-t+1$ benign helpers and the rest $Q-x$ helpers are chosen from $t-2$ malicious helpers (excluding H_j with totally $t-1$ malicious helpers). Last, we know $x \geq t$ and x cannot be greater than Q, we then have in the worst case

$$p_d = \sum_{x=t}^{Q} \frac{\binom{n-t+1}{x} \cdot \binom{t-2}{Q-x}}{\binom{n-1}{Q}} \tag{4}$$

6.3 Message Overhead (N)

We now analyze how many messages P_A and P_B will need to send in one key exchange session with π^A. First, during the Initialization phase, there are two initialization messages (i.e., (INIT, sid) and (INITCONFIRM, sid)), plus n shares of s test keys where every share is a separate message, resulting in $n \cdot s+2$ messages. Then during the Cut-and-choose phase, P_B sends P_A two messages (i.e., ($\mathcal{O}$, $\mathcal{E}$) and (NOTIFY)), and P_A sends P_B a copy of every opening key's every share. With totally $s/2$ opening keys (we assume s is an even number for simplicity) and n shares for each opening key, this leads to $s/2 \cdot n + 2$ messages for this phase. Last, during the Session key derivation phase, P_A sends P_B $s/2$ ciphertexts, plus one final message from P_B for session key verification. Overall, there are $\frac{3n+1}{2}s + 5$ messages in total. i.e.,

$$N = \frac{3n+1}{2}s + 5 \tag{5}$$

From Equation (5), N increases as n and s increase. If a lower message overhead is desired, one can lower the value of n and s (i.e., less helpers and test keys). On the other hand, from Section 6.1, lowering the values of n and s will increase p_f. Therefore, users need to adjust n and s to meet their specific requirements for p_f and N.

7 EXPERIMENTAL RESULTS

7.1 Experiment Design

We implemented π^A with python cryptography libraries and measured its performance, including its running time, CPU cycles, energy consumption, and bandwidth overhead, in experiments.

We set up our experiment devices and running environments as follows. For each key exchange session between a key exchange initiator P_A and a key exchange responder P_B, we selected three different types of resource-constrained devices: Raspberry Pi Zero W, Arduino Due, and SAM D21 Xplained. They are commonly used in the real word for IoT applications but have a different range of resource capacity. Table 1 describes their basic specifications. For the implementations of π^A and other three PKC-based key exchange protocols, we used Python 3.6.9 with cryptography library pycrypto 2.6.1. For the networking environment, we used the Mininet platform [11] on Ubuntu 18.04.4 to emulate a Wi-Fi environment, where every link is 10 Mbps with a 0.02% packet loss probability.

Table 1: Key exchange devices in experiments

	CPU	Memory	Voltage	Current draw
Raspberry Pi Zero W	1 GHZ	512 MB	5 V	500 mA
Arduino Due	84 MHZ	512 KB	1.8 V	77.5 mA
SAM D21 Xplained	48 MHZ	32 KB	1.62 V	7 mA

The main parameters to configure for our experiments are n, t, s, and the number of malicious helpers m. In our experiments, we first set the failure probability of π^A to be 0.005 which was preconfigured by P_A and P_B. With this setup, from Section 6.1, we can derive that π^A has the minimum message overhead when we set n to be 6, t to be 4, and s to be 28. In addition, P_A and P_B can always detect malicious helpers when m is no greater than 2.

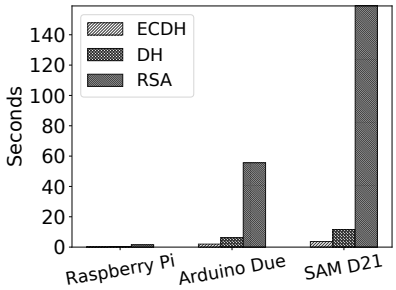
We compare π_6^A with traditional PKC-based key exchange protocols: RSA (Rivest–Shamir–Adleman), DH (Diffie-Hellman), and ECDH (Elliptic Curve Diffie-Hellman). We set the key length of π_6^A to be 128, for which the equivalent key lengths for RSA, Diffie-Hellman, and ECDH are 3072, 3072, and 256, respectively [4]. For ECDH, we use the curve SECP256R1 with ephemeral keys. For each PKC-based protocol, we do not include an authentication component; even so and even as π_6^A includes an authentication (Section 3.1), we show π_6^A outperforms them, many times tremendously.

7.2 Running Time

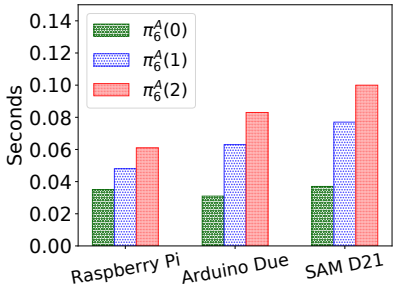
We measured the running time of π_6^A and the comparator key exchange protocols on both P_A and P_B. We recorded the time for running a complete session of each protocol on each device and took the average across 10 experiments. Figure 4 shows the comparison results of π_6^A versus different comparator protocols. Specifically, Figure 4a and Figure 4c show the running time of PKC-based key exchange protocols, while Figure 4b and Figure 4d show the running time of $\pi_6^A(0)$, $\pi_6^A(1)$, and $\pi_6^A(2)$.

Figure 4a and Figure 4b illustrate that on P_A, π_6^A is much faster than its comparators, especially when P_A is an Arduino Due or SAM D21 whose resources are extremely limited. Using $\pi_6^A(2)$ as an example, which has the slowest running time among the three π_6^A configurations in our experiments, on Raspberry Pi Zero W, $\pi_6^A(2)$ in the worst case is 2.3 times faster than ECDH and 24.1 times faster than RSA; however, on SAM D21, $\pi_6^A(2)$ is 59.6 times faster than ECDH and 1591 times faster than RSA.

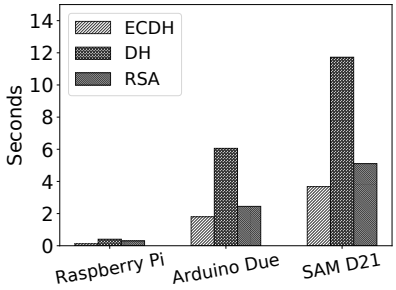
Figure 4c and Figure. 4d show on P_B for all types of IoT devices, although its lead is less striking than that on P_A, π_6^A is still faster



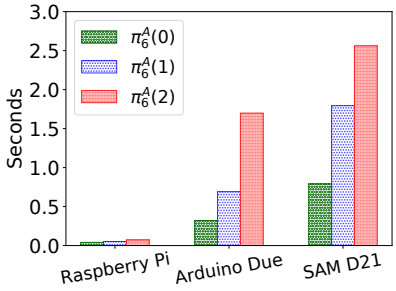
(a) Comparator protocols on device P_A



(b) π_6^A on device P_A



(c) Comparator protocols on device P_B



(d) π_6^A on device P_B

Figure 4: Running time of key exchange protocols on devices P_A and P_B. Note that each subfigure uses a different maximum value for its Y-axis.

than other protocols. Again using $\pi_6^A(2)$ as an example, while on P_A $\pi_6^A(2)$ is 2.3 to 59.6 times faster than ECDH, on P_B it is still about 0.7 to 3.65 times faster than ECDH; with a Raspberry Pi Zero, the running time of $\pi_6^A(2)$ on P_B is 0.072 seconds while it takes ECDH 0.122 seconds. The lead reduction here is because P_B needs to perform more operations than P_A, including identifying malicious helpers, reconstructing evaluation keys, and decrypting multiple ciphertexts to obtain the secret. Nonetheless, π_6^A is faster than its comparator protocols on both devices in a key exchange.

7.3 CPU Cycles

We also measured the CPU cycles of π_6^A and the comparator protocols on both P_A and P_B. As shown in Figure. 5, it takes the comparator protocols many times more CPU cycles than π_6^A to conduct a key exchange session. On P_A, for example, if it is a Raspberry Pi Zero W, it takes ECDH 4.87 times more CPU cycles than $\pi_6^A(2)$ in the worst case, where $\pi_6^A(2)$ is the most expensive among the three different configurations of π_6^A. Similarly, if it is a SAM D21, it takes 4.1 times more instead. On P_B, for example, if it is a Raspberry Pi Zero W, it takes ECDH 11.79 times more CPU cycles than $\pi_6^A(2)$, and if it is a SAM D21, it takes 104.7 times more instead. Again, even though π_6^A's operations on P_B are relatively heavier than P_A, similar to its running time performance on P_B, its CPU cycles on P_B still easily betters those of the comparator protocols.

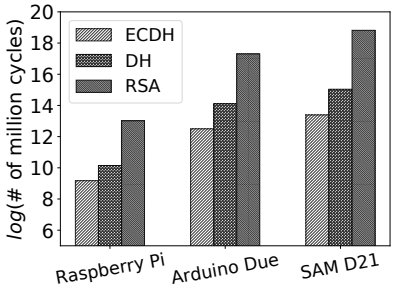
7.4 Energy Consumption

We measured π_6^A and the comparator protocols' energy consumption with the formula $E = U \cdot I \cdot T$ [3] where U is the voltage, I is the current intensity, and T is the time to complete a session of a key exchange protocol. The values of U and I are from Table 1. Notice we only consider the current intensity when devices are in the active mode. Figure 6a and Figure 6b show the energy consumption comparison results at P_A. We can see if P_A is a Raspberry Pi Zero, while the most energy-efficient PKC protocol ECDH consumes 497.5mJ, $\pi_6^A(2)$ in the worst case only consumes 152.5mJ, which is only about 30.6% of ECDH's energy consumption. In fact, the energy saving with π_6^A is even more significant if the device is resource-constrained. For example, if P_A is a SAM D21, while ECDH consumes 42mJ, $\pi_6^A(2)$ only consumes 0.41mJ, which is only 0.97% of ECDH's energy consumption.

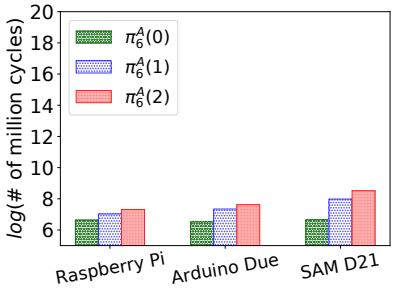
Figure 6c and Figure 6d show energy consumption comparison at P_B. We can see π_6^A again consumes much less energy than the PKC-based key exchange protocols. For example, if P_B is a Raspberry Pi Zero, the energy consumption of $\pi_6^A(2)$ on P_B is 59.1% of that of ECDH (180mJ versus 305mJ), and if P_B is a much more resource-constrained SAM D21, this number becomes 47.9% (20.1mJ versus 41.8mJ). Last, in $\pi_6^A(0)$ and $\pi_6^A(1)$ P_B consumes even less energy than the comparator protocols.

7.5 Bandwidth Overhead

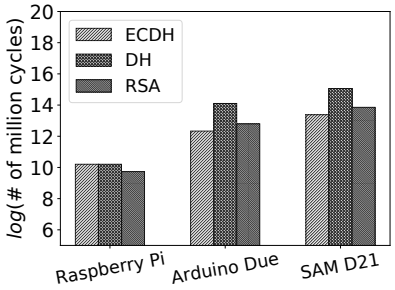
Finally, we measured the bandwidth overhead of π_6^A and its comparator key exchange protocols. In our experiments, the bandwidth overhead indicates the amount of messages that both parties need to transmit over the network in order to establish a session key. Figure 7 illustrates the results. We can see that $\pi_6^A(1)$ and $\pi_6^A(2)$ incur more bandwidth than PKC protocols and $\pi_6^A(0)$ have more bandwidth overhead than ECDH, but less than RSA and Diffie-Hellman. On one hand, the number of messages in π_6^A is much more than that in the other three PKC-based protocols. On the other hand, the length of keys in π_6^A is much shorter (Section 7.1) and the size of messages in π_6^A is much smaller. As a result, overall the bandwidth



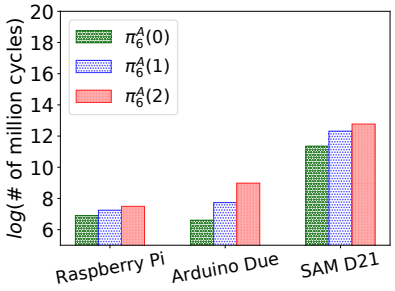
(a) Comparator protocols on device A



(b) π_6^A on device A

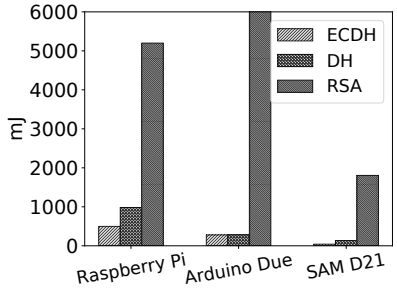


(c) Comparator protocols on device B

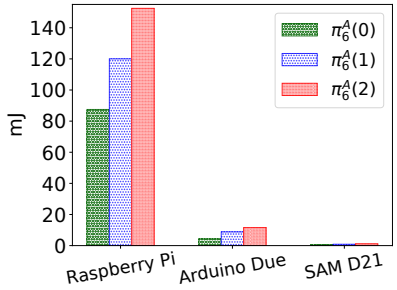


(d) π_6^A on device B

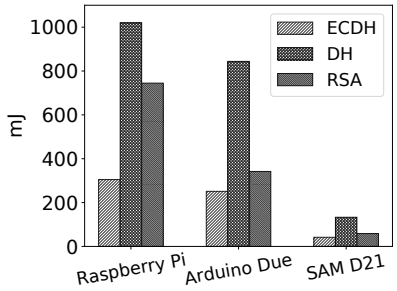
Figure 5: CPU cycles of key exchange protocols on devices P_A and P_B.



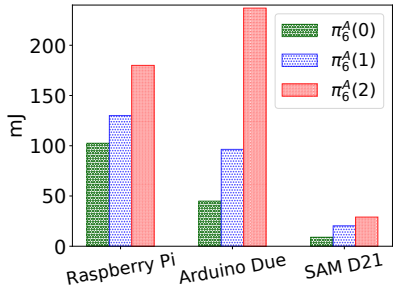
(a) Comparator protocols on device A



(b) π_6^A on device A



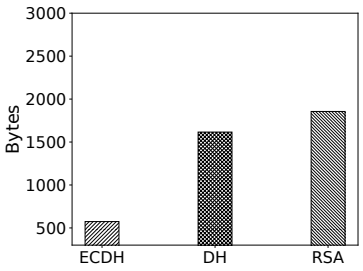
(c) Comparator protocols on device B



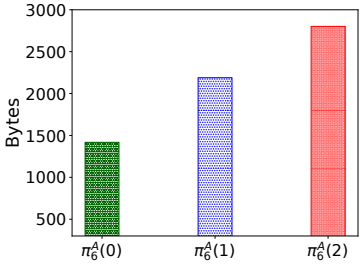
(d) π_6^A on device B

Figure 6: Energy consumptions of key exchange protocols on devices P_A and P_B. Note that each subfigure uses a different maximum value for its Y-axis.

overhead of π_6^A is comparable to that of the comparator protocols, especially when considering its vast improvements in running time and energy consumption. We also emphasize here that the bandwidth overhead in one key exchange session is independent of other key exchange sessions, thus not affected by other sessions. Even if an intermediary may be shared across multiple sessions, it is usually not an IoT device and not poor in bandwidth capacity, further assuring our design is scalable against the size of an IoT network.



(a) Comparator protocols



(b) π_6^A

Figure 7: Bandwidth usage of key exchange protocols.

8 CONCLUSION

Internet of things (IoT) devices have an essential need of secure communications between them, for which a key exchange protocol for them to establish a communication session key is a prerequiste. However, due to their often extremely constrained resources and computing power, many IoT devices are not capable of performing public key cryptography (PKC), making any key exchange solution that uses PKC infeasible. There have been lightweight, non-cryptographic solutions, but they are often unrealistic.

Key exchange solutions that only use symmetric key cryptography (SKC) can be divided into two categories: those using preshared secrets and those using intermediary parties. The former is daunting and hardly scalable when employed for an IoT network composed of hundreds or even thousands of devices. The latter so far relies on honest or semi-honest intermediary parties.

This paper proposes a new SKC-based key exchange solution (π^A) using intermediary parties (also called helpers). It departs from the state of the art by assuming any intermediary party can be malicious. Its design makes it lightweight and deployable in IoT and resilient against malicious intermediary parties. In particular, under the cut-and-choose methodology, π^A introduces a new protocol design that not only can successfully establish a session key in the end, but also can efficiently identify malicious intermediary parties when they tamper messages going through them, even if they collude or selectively tamper messages.

This paper provided both theoretical proof and analysis and empirical evaluations of π^A. From the proof π^A is shown to be secure against malicious helpers. From the analysis, π^A's failure probability is easily negligible with a reasonable setup and π^A's malicious helper detection probability can be 1.0 even when a malicious helper only tampers a small number of messages. From the empirical evaluations, π^A outperforms three widely used PKC-based key exchange protocols in terms of running time, CPU cycles, and energy consumption while its bandwidth overhead is comparable to them.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable feedback. This material is based upon work partially supported by the Ripple Graduate Fellowship awarded to Zhangxiang Hu. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the supporters.

REFERENCES

[1] Arash Afshar, Zhangxiang Hu, Payman Mohassel, and Mike Rosulek. 2015. How to Efficiently Evaluate RAM Programs with Malicious Security. In *Advances in Cryptology – EUROCRYPT 2015*. 702–729.
[2] Haithem Al-Mefleh and Osameh Al-Kofahi. 2016. Taking advantage of jamming in wireless networks: A survey. *Computer Networks* 99 (2016), 99 – 124.
[3] Giuseppe Ateniese, Giuseppe Bianchi, Angelo Capossele, and Chiara Petrioli. 2013. Low-cost Standard Signatures in Wireless Sensor Networks: A Case for Reviving Pre-computation Techniques?. In *NDSS*.
[4] Elaine Barker, William Burr, William Polk, and Miles Smid. 2006. *Recommendation for key management: Part 1: General*. National Institute of Standards and Technology, Technology Administration.
[5] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. 1998. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. Cryptology ePrint Archive. https://eprint.iacr.org/1998/009.
[6] Daniel J. Bernstein. 2006. Curve25519: New Diffie-Hellman Speed Records. In *Public Key Cryptography*. Berlin, Heidelberg, 207–228.
[7] Ran Canetti and Hugo Krawczyk. 2001. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. Cryptology ePrint Archive, Report 2001/040. https://eprint.iacr.org/2001/040.
[8] Ran Canetti and Hugo Krawczyk. 2002. Universally Composable Notions of Key Exchange and Secure Channels. Cryptology ePrint Archive, Report 2002/059. https://eprint.iacr.org/2002/059.
[9] Craig Costello and Patrick Longa. 2015. FourQ: Four-Dimensional Decompositions on a Q-curve over the Mersenne Prime. In *Advances in Cryptology*. Berlin, Heidelberg, 214–235.
[10] Mario Di Raimondo and Rosario Gennaro. 2003. Provably Secure Threshold Password-Authenticated Key Exchange. In *Advances in Cryptology*. Berlin, Heidelberg, 507–523.
[11] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. 2012. Reproducible Network Experiments Using Container-Based Emulation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*. New York, USA, 253–264.
[12] Haowen Chan, A. Perrig, and D. Song. 2003. Random key predistribution schemes for sensor Networks. In *Symposium on Security and Privacy*. 197–213.
[13] R. Hummen, H. Shafagh, S. Raza, T. Voig, and K. Wehrle. 2014. Delegation-based authentication and authorization for the IP-based Internet of Things. In *Eleventh Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. 284–292.
[14] M. A. Iqbal and M. Bayoumi. 2016. Secure End-to-End key establishment protocol for resource-constrained healthcare sensors in the context of IoT. In *International Conference on High Performance Computing Simulation (HPCS)*. 523–530.
[15] Bocheng Lai, Sungha Kim, and Ingrid Verbauwhede. 2002. Scalable Session Key Construction Protocol for Wireless Sensor Networks. In *IEEE Workshop on Large Scale RealTime and Embedded Systems (LARTES)*.
[16] Sang-Gi Lee, Sei-Yoon Lee, and Jeong-Chul Kim. 2016. A Study on Security Vulnerability Management in Electric Power Industry IoT. *Journal of Digital Contents Society* 17, 6 (2016), 499–507.
[17] X. Liang, R. Peterson, and D. Kotz. 2020. Securely Connecting Wearables to Ambient Displays with User Intent. *Transactions on Dependable and Secure Computing* 17, 4 (2020), 676–690.
[18] Yehuda Lindell and Benny Pinkas. 2007. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In *Proceedings of the 26th Annual International Conference on Advances in Cryptology*. Berlin, Heidelberg, 52–78.
[19] Donggang Liu and Peng Ning. 2003. Establishing Pairwise Keys in Distributed Sensor Networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)* (Washington D.C., USA). New York, USA, 10 pages. https://doi.org/10.1145/948109.948119
[20] P. MacKenzie, Thomas Shrimpton, and M. Jakobsson. 2002. Threshold password-authenticated key exchange: (Extended abstract). In *CRYPTO*.

[21] Muslum Ozgur Ozmen and Attila A. Yavuz. 2017. Low-Cost Standard Public Key Cryptography Services for Wireless IoT Systems. In *Proceedings of the Workshop on Internet of Things Security and Privacy*. 65–70.

[22] Timothy J. Pierson, Travis Peters, Ronald Peterson, and David Kotz. 2019. CloseTalker: Secure, Short-Range Ad Hoc Wireless Communication. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services* (Seoul, Republic of Korea) *((MobiSys))*. 340–352. https://doi.org/10.1145/3307334.3326100

[23] P. Porambage, A. Braeken, A. Gurtov, M. Ylianttila, and S. Spinsante. 2015. Secure end-to-end communication for constrained devices in IoT-March-enabled Ambient Assisted Living systems. In *2nd World Forum on Internet of Things (WF-IoT)*. 711–714.

[24] P. Porambage, A. Braeken, P. Kumar, A. Gurtov, and M. Ylianttila. 2015. Proxy-based end-to-end key establishment protocol for the Internet of Things. In *International Conference on Communication Workshop (ICCW)*. 2677–2682.

[25] Y. B. Saied and A. Olivereau. 2012. D-HIP: A distributed key exchange scheme for HIP-based Internet of Things. In *International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. 1–7.

[26] Yogeesh Seralathan, Tae Tom Oh, Suyash Jadhav, Jonathan Myers, Jaehoon Paul Jeong, Young Ho Kim, and Jeong Neyo Kim. 2018. IoT security vulnerability: A case study of a Web camera. In *20th International Conference on Advanced Communication Technology (ICACT)*. 172–177.

[27] Stefaan Seys and Bart Preneel. 2002. Key establishment and authentication suite to counter DoS attacks in distributed sensor networks. *Unpublished manuscript). COSIC* (2002).

[28] Adi Shamir. 1979. How to Share a Secret. *Communication* 22, 11 (November 1979), 612–613.

[29] J. Zhang, Z. Wang, Z. Yang, and Q. Zhang. 2017. Proximity based IoT device authentication. In *Conference on Computer Communications*. 1–9.

[30] Zhi-Kai Zhang, Michael Cheng Yi Cho, Chia-Wei Wang, Chia-Wei Hsu, Chong-Kuan Chen, and Shiuhpyng Shieh. 2014. IoT security: ongoing challenges and research opportunities. In *IEEE 7th international conference on service-oriented computing and applications*. 230–234.