Attestation with Constrained Relying Party

Mariam Moustafa[2,3][a], Arto Niemi[1][b], Philip Ginzboorg[1][c] and Jan-Erik Ekberg[1]

[1]*Huawei Technologies Oy, Helsinki, Finland*

[2]*Aalto University, Espoo, Finland*

[3]*Denmark Technical University, Copenhagen, Denmark*

Keywords: Remote Attestation, Device Security, Model Checking.

Abstract: Allowing a compromised device to e.g., receive privacy-sensitive sensor readings carries significant privacy risks, but to implement the relying party of a contemporary attestation protocol in a computationally constrained sensor is not feasible, and the network reach of a sensor is often limited. In this paper, we present a remote platform attestation protocol suitable for relying parties that are limited to symmetric-key cryptography and a single communication channel. We validate its security with the ProVerif model checker.

1 INTRODUCTION

Remote attestation allows a device's security state to be validated remotely (Coker et al., 2011). The attestation evidence is remotely appraised by a verification service, typically by matching measurements reported in the evidence against trusted reference values. Securely transmitting the attestation challenge, evidence and results is the task of a remote attestation protocol. The protocol must provide at least message integrity, freshness and origin authentication, as well as resistance against relay attacks.

Remote attestation is today deployed commercially on servers and high-end consumer devices such as smartphones and PCs (Niemi et al., 2023). Standardization is also progressing, holding promise of interoperability in the future (Birkholz et al., 2023). On constrained devices, remote attestation is still rarely used, despite an abundance of proposals from both academia (Johnson et al., 2021) and industry (TCG, 2021; Hristozov et al., 2022). This stems from the constrained devices' lack of computing power and memory, which makes them incapable of performing the public-key cryptography required by most remote attestation protocols. Proposed protocols have focused on enabling powerful devices to validate the security of constrained ones.

In this paper, we consider the reverse situation: how can a constrained device confirm the security state of the system it is interacting with? A sensor may measure data that reveals information of the user's illnesses or lifestyle: releasing such information to another device without first validating the integrity of the receiving software may result in a privacy violation. Similarly, a key tag should not transfer key material to a malware-infested smartphone. In such settings, the constrained device operates as the relying party in attestation protocol – a visualization of this is shown in Fig. 1.

The IETF RFC 7228 standard (Bormann et al., 2014) defines a sensor-class device to have less than 10 KB data and less than 100 KB code; too little for asymmetric cryptography even if performance and battery consumption concerns are ignored. We work under the assumptions that the relying party device a) does not support public-key cryptography; and b) can only communicate with an external attestation verifier via the attester, which makes man-in-the-middle attacks on protocol messages a specific concern.

We present to our knowledge the first remote attestation protocol with the following contributions:

1. We analyze a less-considered attestation flow, where constrained devices need to attest a system before becoming part of its operation.
2. We provide an attestation protocol design where the relying party does not need public-key cryptography and can participate via a single insecure channel it has with the attester.
3. We validate the security of our protocol using formal model checking and provide a working proof-of-concept, whose performance metrics confirm the protocol's viability in practice.

[a] https://orcid.org/0009-0003-3046-8675

[b] https://orcid.org/0000-0003-3118-4511

[c] https://orcid.org/0000-0003-4579-3668

Moustafa, M., Niemi, A., Ginzboorg, P. and Ekberg, J.
Attestation with Constrained Relying Party.
DOI: 10.5220/0012319300003648

In *Proceedings of the 10th International Conference on Information Systems Security and Privacy (ICISSP 2024)*, pages 701-708
ISBN: 978-989-758-683-5; ISSN: 2184-4356

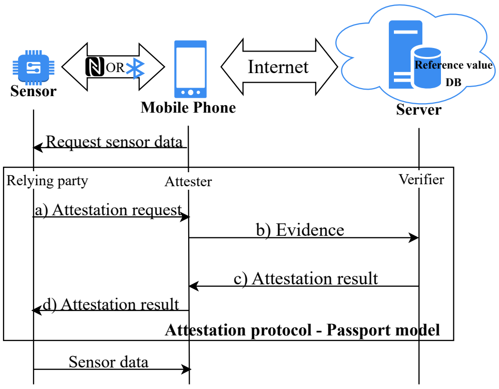


Figure 1: A constrained sensor operating as a relying party: The security of a mobile phone is verified as a precondition for transmitting sensor readings. Note the absence of direct communication between RP and verifier.

2 BACKGROUND

2.1 Remote Attestation

Following the IETF and Trusted Computing Group attestation architectures (Birkholz et al., 2023; TCG, 2021), remote attestation protocols involve three active participants: *attester*, *verifier* and *relying party* (RP). The RP wants to know the state of the attester before making a trust decision, such as granting access to a resource. The attester is equipped with a trustworthy mechanism (Coker et al., 2011) such as a Trusted Execution Environment (TEE), (Gunn et al., 2022) which collects and cryptographically protects attestation evidence. The evidence may, for example, include boot-time code measurements or whether the current user has been authenticated. The *verifier* appraises the evidence and issues a verdict (attestation results) to the *relying party*.

Two interaction patterns are commonly used in remote attestation: the background check and the passport model (Birkholz et al., 2023). The latter, illustrated in Fig. 1, is better suited to resource-constrained RP, as it requires only a single communication link in the RP. The attester carries the attestation results produced by the verifier as a “passport”, which the attester stores and then presents to the RP when needed. The attester may also obtain fresh results from the verifier if the current results are older than what the RP accepts.

2.2 Related Work

The SlimIoT protocol (Ammar et al., 2018) performs swarm attestation where the entities being attested are a ‘swarm’ of constrained IoT devices. The opetation is based on broadcast and selective evidence aggregation among attesters. SCAPI (Kohnhäuser et al., 2017) is another swarm-based attestation protocol, leveraging a Trusted Execution Environment (TEE) for tamper-resistance. Additionally each attester has shared keys with all other attesters in the network which adds memory footprint and power consumption overhead. Jäger et al. propose a protocol (Jäger et al., 2017) where the server sends a nonce as a challenge, the attester hashes the nonce with the key being attested, and the verifier checks that the hash value is as intended – limiting the attestation result to a key possession argument. AAoT (Feng et al., 2018) is based on physical unclonable functions (PUFs) to generate the symmetric keys between the attester and verifier. In the SIMPLE protocol (Ammar et al., 2020), the verifier generates a nonce, the value of a valid software state, and the MAC tag of these values. The attester verifies the tag, computes its own state, and checks whether the computed state matches what the verifier sent.

These protocols only cater to the case where the attester is the constrained device. This is evidenced by the lack of separation between verifier and RP roles. Also, some of the protocols require synchronized clocks, or special hardware, like PUFs, to be present in the constrained device.

3 REQUIREMENTS

3.1 Functional Requirements

To ensure the viability of our protocol in the use case where the relying party (RP) is a constrained device, we require the following:

FR1. The RP can be implemented with $<$ 10 KB of code, including cryptography, but excluding the transport protocol such as UDP or Bluetooth.

FR2. The RP does not need public-key cryptography.

FR3. The RP only needs to communicate with the attester.

3.2 Security Requirements

We assume the Dolev-Yao attacker model: the attacker can read, intercept, insert, relay and modify protocol messages, but is not able to guess secret keys or to break cryptographic primitives (Dolev and Yao, 1983). The attacker can use uncompromised devices as oracles and pretend to be any of the participating entities, but is not able to compromise the verifier or the TEE of the attester.

The attacker's goal is to trick an uncompromised RP into trusting a compromised device. The uncompromised RP is assumed to execute its part of the protocol correctly, so it will only trust an attester after receiving a valid attestation result that the RP believes to describe the security state of that particular attester. Thus, we formulate our security requirements in terms of the security of the attestation results:

SR1. Freshness of attestation results: the RP can detect whether an attestation result was generated in a particular run of the protocol.

SR2. Binding of attestation results to a particular attester: the RP can detect whether the verifier generated the result based on its appraisal of a particular attester.

SR3. Integrity of attestation results: the RP can detect whether an attestation result has been generated by a particular verifier, and whether the result has been modified in transit.

SR4. Confidentiality of attestation metrics and attestation results: the attester measurements and results are encrypted to ensure privacy.

4 PROTOCOL DESIGN

Our protocol uses the passport model discussed in Section 2, with the provision that attestation results cannot be reused and are bound to a particular relying party. Accordingly, we call our protocol Attestation Protocol for Constrained Relying Parties – Live Passport Model, or APCR-LPM for short.

Fig. 2 illustrates the protocol steps. For symmetric encryption (denoted *senc* and *sdec*), we use a cipher that provides authenticated encryption, such AES-CCM. Thus the *sdec* operation either returns the decrypted plaintext or, if integrity violation was detected, an error. Similarly, signature validation ($checksig(sig, m, PK)$) either returns the signed message or an error. The *aenc*, and *adec* are encryption and decryption operations of a public-key authenticated encryption scheme, e.g., ECIES (3GPP, 2023).

The protocol involves three principals:

- Relying Party RP: a constrained device with little memory, supporting only symmetric-key cryptography. It has a communication link with the attester and can verify attestation results, but not attestation evidence.
- Attester A: a non-constrained device, such as a smartphone, that wants to prove its trustworthiness to the relying party. It has a trustworthy mechanism for evidence generation and secure storage for secrets, such as a TEE.
- Verifier V: a non-constrained device, such as a cloud server, trusted by both RP and attester. It can validate and appraise the evidence sent by the attester and is assumed to have agreed upon an evidence appraisal policy with the RP.

Bootstrapping. At the start of the protocol, the relying party (RP) has symmetric keys K_A, and K_V, shared with the attester and verifier, respectively. We envision three possible key distribution scenarios:

1. The relying party (e.g., wearable, smart device) and attester belong to the same vendor. In these cases, the key material can be installed in these devices during manufacturing.
2. A user-based bootstrapping or pairing protocol involving an out-of-band channel has been executed by the owner of the devices, which results in a shared key between devices.
3. The relying party (sensor) has a predefined relationship with the verifier. The verifier also takes on the role of a key distribution center that creates and distributes session keys to the relying party and attester.

The attester has an asymmetric keypair (SK_A, PK_A) that is uses to authenticate itself to the verifier. The verifier is assumed to either trust PK_A directly or able to construct a trust chain, e.g. with X.509 certificates, that allows it to trust PK_A. The RP also has an identifier id_A for the attester A that is a function of both K_A and PK_A.

The protocol steps, shown in Fig. 2 are as follows: (1) The relying party (RP) prepares a challenge by generating the nonce c. It encrypts c and the identifier id_A of the attester using K_V (2). The nonce c acts as a session identifier as well as a freshness value. RP sends the challenge to the attester A, message (a). The attester cannot read the contents of the message as it is encrypted with a key that it does not know.

The attester's TEE performs key attestation on the hash of K_A (4). By including the key attestation, the TEE vouches that K_A cannot be extracted from the TEE. In step (5), the attester collects the attestation metrics of the device and its software. Then, it encrypts the key attestation, collected metrics, and challenge it received from RP using PK_V (6), signs the encrypted evidence using its private key SK_A (7), and finally sends the evidence and signature to the verifier in message (b).

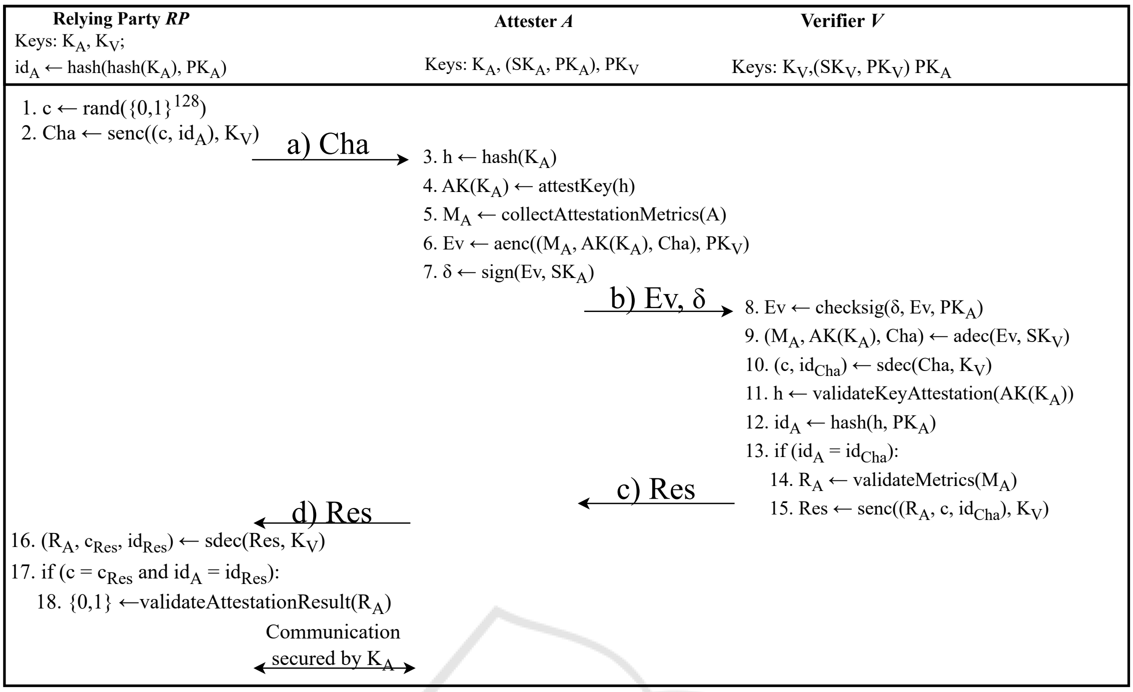

Relying Party *RP*
Keys: K_A, K_V;
id_A ← hash(hash(K_A), PK_A)
Attester *A*
Keys: K_A, (SK_A, PK_A), PK_V
Verifier *V*
Keys: K_V,(SK_V, PK_V) PK_A
1. c ← rand($\{0,1\}^{128}$)
2. Cha ← senc((c, id_A), K_V)
a) Cha
3. h ← hash(K_A)
4. AK(K_A) ← attestKey(h)
5. M_A ← collectAttestationMetrics(A)
6. Ev ← aenc((M_A, AK(K_A), Cha), PK_V)
7. δ ← sign(Ev, SK_A)
b) Ev, δ
8. Ev ← checksig(δ, Ev, PK_A)
9. (M_A, AK(K_A), Cha) ← adec(Ev, SK_V)
10. (c, id_{Cha}) ← sdec(Cha, K_V)
11. h ← validateKeyAttestation(AK(K_A))
12. id_A ← hash(h, PK_A)
13. if (id_A = id_{Cha}):
14. R_A ← validateMetrics(M_A)
15. Res ← senc((R_A, c, id_{Cha}), K_V)
c) Res
d) Res
16. (R_A, c_{Res}, id_{Res}) ← sdec(Res, K_V)
17. if (c = c_{Res} and id_A = id_{Res}):
18. {0,1} ←validateAttestationResult(R_A)
Communication secured by K_A

Figure 2: Attestation Protocol for Constrained Relying Party.

Table 1: Summary of notation.

Term	Description
K_A	Shared symmetric key between *A* and *RP*.
K_V	Shared symmetric key between *V* and *RP*.
(SK_V, PK_V)	The (secret key, public key) pair of *V*.
(SK_A, PK_A)	The (secret key, public key) pair of *A*.
h	Hash of K_A.
c	A 128-bit pseudorandom value.
M_A	The attestation metrics produced by *A*.
$r \leftarrow rand(\{0,1\}^{128})$	Generate pseudorandom 128-bit string.
$c \leftarrow senc(m,K)$	Authenticated encryption of *m* with shared key *K*.
$m \leftarrow sdec(c,K)$	Authenticated decryption of *c* with shared key *K*.
$c \leftarrow aenc(m,PK)$	Public-key authenticated encryption of *m* with public key *PK*.
$m \leftarrow adec(c,SK)$	Public-key authenticated decryption of *c* with secret key *SK*.
$sig \leftarrow sign(m,SK)$	Signing of *m* with secret key *SK*.
$m \leftarrow checksig(sig,m,PK)$	Verifying the signature of *m* with key *PK*.
$h \leftarrow hash(m)$	Computing the hash of *m*.
$AK(K) \leftarrow attestKey(h)$	Attestation of key *K* by TEE.
$h \leftarrow validateKeyAttestation(AK(K))$	Validate that key *K* is attested by TEE.
$M \leftarrow collectAttestationMetrics(E)$	Compute the attestation metrics of entity *E*.
$R \leftarrow validateMetrics(M)$	Compute attestation results *R* based on metrics *M*.
$\{0,1\} \leftarrow validateAttestationResult(R)$	Determine trustworthiness based on attestation results *R*.

The verifier verifies the signature of the evidence using PK_A (8) and decrypts the evidence with SK_V. Then it decrypts the challenge to extract *c* and *h* (10). In step (11), it verifies the key attestation and computes its own version of the attester's id_A based on the public key it has used for checking the signature (12). In step (13), the verifier checks whether id_A received in the challenge is equal to that it has computed. If any of those steps fail, the verifier aborts the protocol. Based on the metrics, the verifier generates a verdict or attestation result for RP (14) and encrypts the attestation results together with the nonce *c* and id_{Cha} using K_V (15) resulting in the ciphertext *Res*. The verifier sends *Res* to the attester in message (c). The attester forwards *Res* to RP in message (d). The value id_{Cha} is included in Res as an indicator to RP that the verifier has verified the evidence of the attester with identity id_{Cha}. It also includes the decrypted challenge (c, id_{Cha}) to maintain the freshness of the message and to indicate to RP that the intended verifier has received the challenge and decrypted it.

(16) RP decrypts *Res* and checks that the values of c_{Res} and id_{Res} are equal to the ones it sent in the challenge (17). RP can then process the attestation result to determine the attester's state (18).

Subsequent, application-specific communication between RP and attester depends on the result of step (18); this communication is secured using K_A.

5 SECURITY ANALYSIS

5.1 Discussion

APCR-LPM fulfills the security requirements of Section 3.2 as follows:

SR1 (Freshness of Attestation Results). The relying party (RP) includes a nonce c in the challenge and the verifier is required to include the same nonce in the attestation results. The nonce is a pseudorandom number that is generated fresh in every run of the protocol. In step (17), the RP checks whether the nonce in the received attestation result matches the nonce it sent in the last challenge. For a replayed result, the check will fail. The RP could also use a protocol timeout to prevent the acceptance of obsolete results.

SR2 (Binding of Attestation Results to a Particular Attester). The RP binds the challenge Cha to the identity of a particular attester by including the value $id_A = hash(hash(K_A), PK_A)$. The verifier knows PK_A, the public key of the attester's TEE, and receives $hash(K_A)$ from the key attestation included in the attestation evidence, so it can compute a reference id_A. By verifying the evidence signature with PK_A in step (8) and by comparing the self-computed id_A against the one decrypted from Cha in step (13), the verifier can detect whether the evidence was generated by a TEE that the verifier trusts and that belongs to the attester the RP intended. Since we assume the evidence signing keys (SK_A) to be unique to the TEE instance and unextractable, the verifier can validate that the evidence was generated by the particular TEE that is identified with PK_A. The verifier includes c and the validated id_A in the results, allowing the RP to check, in step (17), that they match the values it sent in the challenge. This fulfills the requirement, preventing relay attacks, sometimes called Cuckoo attacks (Parno, 2008; Dhar et al., 2020), a common issue with remote attestation protocols (Niemi et al., 2021; Aldoseri et al., 2023).

SR3 (Integrity of Attestation Results). The RP checks the integrity of the attestation result by decrypting, in step (16), the result message Res with the shared key (K_V) it has with the verifier. Since Res is protected with authenticated encryption, using a key (K_V) that the attacker does not know, the RP can detect whether the message was modified after encryption or generated by a different verifier. This fulfills the integrity requirement. Finally, in step (18), the RP can evaluate the trustworthiness of the attester it identified in the challenge by examining the verifier's verdict that it decrypts from Res.

SR4 (Confidentiality of Attestation Metrics and Attestation Results). The confidentiality of the attestation metric is guaranteed with public key cryptography, where the evidence is encrypted using the verifier's public key PK_V in step (6). Only the verifier can read the evidence using its private key SK_V. The attestation result, on the other hand, is encrypted using the symmetric key K_V the verifier shares with the relying party, step (15). Only the parties who know K_V can read the attestation results.

5.2 Formal Model Checking

We used the ProVerif tool (Blanchet et al., 2018) to formally model our protocol and verify its security properties. Queries describing the desired security properties are included in the ProVerif model. These queries are written in terms of ProVerif events which mark certain stages reached by the protocol and have no effect on the actual behavior of the model. The tool attempts to explore all possible execution paths of the protocol, trying to find a path where a query fails. ProVerif assumes the Dolev-Yao attacker model and can perform replay, man-in-the-middle and spoofing (impersonation) attacks, which aligns with the adversary model in Section 3.2. Our ProVerif code is available in GitHub [1].

The following query represents the security requirements SR1, SR2, and SR3 in Section 3.2:

$$\begin{aligned}
&query\ PK_A : pkey, K_A : key, K_V : key, R_A : bitstring, c : nonce,\\
&h : bitstring, id : bitstring, M_A : bitstring, Cha : bitstring;\\
&inj-event(relyingPartyAccepts(K_V, R_A, c, id))\\
&\quad \Longrightarrow\ inj-event(relyingPartyBegins(K_V, c, id))\ \&\&\\
&\qquad inj-event(attesterBegins(PK_A, h, M_A, Cha))\ \&\&\\
&\qquad inj-event(verifierAccepts(PK_A, K_V, M_A, id, c))\ \&\&\\
&\qquad R_A = validateEvidence(M_A)\ \&\&\\
&\qquad id = hash((h, PK_A))\ \&\&\\
&\qquad Cha = senc((c, id), K_V).
\end{aligned}$$

The query defines an injective correspondence between the event of the relying party (RP) accepting R_A and all other events in the protocol. This means that for each occurrence of the event $relyingPartyAccepts$ there is a distinct occurrence of all other events in the query. The RP will only accept the protocol run if there has been a previous run where it:

1. Initiated the protocol by sending the encrypted c and id ($relyingPartyBegins$);
2. The attester has accepted this encrypted c and id as Cha and collected attestation metrics M_A ($attesterBegins$);
3. The verifier has received M_A and decrypted Cha ($verifierAccepts$).

There is an added constraint on the relation between the metrics M_A and the attestation results R_A. This query models the security requirements (Section 3.2) by including c in the events for freshness, the keys for data origin authentication and M_A and R_A for integrity. M_A is matched to both the attester and verifier event

[1] https://github.com/Mariam-Dessouki/RAforConstrainedRP/tree/apcr-paper

and the RP will not accept the final message unless R_A is a function of M_A and it has received back the encrypted *c* and *id* it used in the first message.

In order to satisfy SR1 (freshness of attestation results), the RP subprocess simulates the protocol by generating a new nonce *c* with every protocol run.

When the RP receives the value *Res*, it does the following.

```
let (R_a:bitstring, =c , =id)=sdec(Res, K_v) in
event relyingPartyAccepts(K_v, R_a, c, id);
```

The RP first checks that the nonce *c* is the same as the one it has sent and then it invokes the event *relyingPartyAccepts*. The equal sign (before *c* and *id*) matches the decrypted value to an already defined value. If the received nonce does not match it an error would occur and the event would not be invoked. For SR3 (integrity of attestation result), the RP checks the integrity of the received *Res* by using the key K_V it shares with the verifier to decrypt the message. If the values of the received *c* and id are not equal to the sent values then the event will not be invoked.

The following code checks the binding of the attestation result to a particular attester (SR2). It is executed after the verifier checks the signature of the evidence.

```
let id = hash((h, PK_a)) in
if (id = id_cha) then
 let R_a = validateEvidence(M) in
 event verifierAccepts(PK_a, K_v, M, id, c);
```

The event $verifierAccepts$ is invoked after the verifier subprocess checks that the *id* value received from the RP matches the *id* value it generated from the attester's public key, otherwise $verifierAccepts$ would not occur.

The query would fail if any of the security requirements are not satisfied. ProVerif was able to check all possible protocol states and terminate. It did not find an attack against the query defined above, i.e, the security requirements SR1, SR2 and SR3 are satisfied.

The query used to represent the security requirement SR4 is as follows:

$$query\ K_V : key, id : bitstring, R_A : bitstring, c : nonce;$$
$$attacker(R_A)\ \&\&\ event(relyingPartyAccepts(K_V, R_A, c, id))$$
$$\implies false.$$

The query states that the events of the attacker knowing the attestation result R_A and the RP accepting the same attestation result cannot occur together. Note that although it is not explicitly mentioned in the query, the query also includes the secrecy of the attestation metrics M_A. The attestation results R_A is a function of M_A, so even if R_A is encrypted, an attacker that knows M_A can easily derive R_A using the function $validateMetrics$ (see (14) in Fig. 2). ProVerif did not find an attack on this query meaning that the attestation metrics and results are confidential, thus satisfying SR4.

6 PROOF-OF-CONCEPT

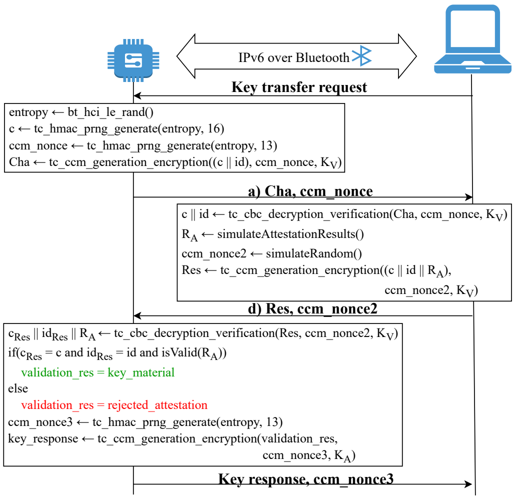
To study the feasibility of our protocol on constrained devices, we implemented the relying party (RP) role on the nRF5340 development kit (Nordic Semiconductor, 2023). The RP was set up to communicate with a laptop over Bluetooth. Since the main advantage of the protocol is that it is suitable for constrained RPs, only the communication between RP and the attester (messages (a) and (d)) and the actions in the relying party (steps (1, 2, 16, 17, 18)) found in Fig. 2 were implemented.

As an example use case we took the electronic lock-and-key system, where the user wants to use his mobile phone (instead of a NFC or Bluetooth keytag) to open doors — in essence copying the key material normally residing on the keytag. The computationally weak keytag needs a way to attest the security of a mobile phone before releasing any cryptographic secrets to it. Our attestation protocol is applicable to this setup as follows. The keytag (relying party) initiates the attestation protocol over a near-field communication channel with the smartphone (attester), who in turn relies on a network server (verifier) to prove the security level of the TEE in the smartphone. After successful verification of attestation results, the keytag transfers the door key to the smartphone TEE.

6.1 Implementation

The nRF5340 board has a 64 MHz Arm Cortex-M33 CPU, 256 KB Flash and 64 KB RAM. It supports the Zephyr RTOS (Real-Time Operating System), an operating system designed for resource constrained devices. We extended Zephyr's sample Bluetooth application, called IPSP (Zephyr, 2023), with the implementation of the key transfer protocol shown in Fig. 3. For cryptography, we used AEC-128-CCM, HMAC-SHA256 and PRNG from the TinyCrypt library (Intel, 2017).

As illustrated in Fig. 3, after the Bluetooth connection is established, the laptop (attester) issues a key transfer request which initiates the ACPR-LPM protocol. The keytag (RP) generates a nonce and encrypts both the nonce *c* and *id* using K_V and AES-CCM. The resulting ciphertext, *Cha*, is sent over the channel in message (a). Upon receiving that message, the application decrypts *Cha* to extract *c* and id. Next, the application encodes the attestation re-



```
entropy ← bt_hci_le_rand()
c ← tc_hmac_prng_generate(entropy, 16)
ccm_nonce ← tc_hmac_prng_generate(entropy, 13)
Cha ← tc_ccm_generation_encryption((c || id), ccm_nonce, $K_V$)
```

```
c || id ← tc_cbc_decryption_verification(Cha, ccm_nonce, $K_V$)
$R_A$ ← simulateAttestationResults()
ccm_nonce2 ← simulateRandom()
Res ← tc_ccm_generation_encryption((c || id || $R_A$),
                                    ccm_nonce2, $K_V$)
```

```
$c_{Res}$ || $id_{Res}$ || $R_A$ ← tc_cbc_decryption_verification(Res, ccm_nonce2, $K_V$)
if($c_{Res}$ = c and $id_{Res}$ = id and isValid($R_A$))
   validation_res = key_material
else
   validation_res = rejected_attestation
ccm_nonce3 ← tc_hmac_prng_generate(entropy, 13)
key_response ← tc_ccm_generation_encryption(validation_res,
                                            ccm_nonce3, $K_A$)
```

Figure 3: Prototype implementation of keytag application.

sult and sends the encrypted c, id, and R_A over the Bluetooth channel. For attestation result, we used the Entity attestation token Attestation Result (EAR) (Fossati et al., 2023) emerging standard. We encoded the EAR object using CBOR (Bormann and Hoffman, 2020) to minimize message and decoder size. Depending on the Res received in message (d), the keytag may decide to send the key material or not. In either case, the response has the same message size to mitigate side channel attacks.

6.2 Measurements

Compared to the original IPSP application, the RP application requires 6 KB more flash and less than 1 KB more RAM memory (Table 2). These figures, based on the values reported by Zephyr at build-time, include protocol processing, code from the TinyCrypt and zcbor libraries, and code specific to the keytag application. To exclude Zephyr from the measurements, we also directly measured the code size of the binaries using the size utility. The RP implementation, including protocol processing and code that calls TinyCrypt or zcbor, required around 2.6 KB of extra code in the binaries.

Table 2: Memory footprint of applications (in bytes).

Memory Type	IPSP Sample	RP Application
Total Flash	241320	247320 (+6000)
Total RAM	60280	61184 (+904)
Application Code Size	1792	4396 (+2604)

Overall, the "keytag" transfers 174 bytes: 1) 55 bytes Cha, which includes 16 bytes c, 16 bytes id, 13 bytes AES-CCM nonce, and 10 bytes MAC; and 2) 119 bytes encrypted key material in the last message.

Reducing Cha size to 16 bytes (one AES block) and analyzing the security implications is left for the future. The "keytag" receives 194 bytes: 20 bytes key transfer request in the first message, and 174 bytes encrypted attestation results in message (d). To test the time overhead of APCR-LPM, we did three experiments, described below. In each experiment the timing of operations was repeatedly measured 10 times on the attester (laptop) side.

1. Baseline Communication Cost. We measured the time between attester sending a key request and it receiving a key response from the RP, without attestation messages (a) and (d), and without cryptographic operations in the RP. This takes 93 ms on the average.
2. Protocol Cost. We recorded the elapsed times (i) from when the attester sends a key transfer request to when it receives message (a), and (ii) from sending message (d) to it receiving key response and the ccm_nonce3. The time required for the protocol steps in the attester was omitted. The sum of (i) and (ii), which is 289 ms on average, measures the cost of overall protocol communication and the Cost of protocol processing in the resource-constrained RP device.
3. Protocol Communication Cost. We sent the same number and size of messages as in experiment (2), but omitted other processing such as message decoding and cryptography. The result was 281 ms on average, 7 ms less than the full protocol cost.

We conclude that APCR-LPM has a time overhead of about 200 ms, compared to insecure communication with the RP. Most of the overhead is due to the additional round-trip, rather than encryption, decryption, and attestation result parsing in the RP.

7 CONCLUSIONS

In this paper, we presented the design of APCR-LPM, an attestation protocol that addresses the neglected use case where the beneficiary of attestation, the relying party, is a constrained device, capable only of symmetric key operations and able to communicate only with the device whose trustworthiness it wants to evaluate. We established the protocol's security via analysis and model checking using ProVerif. Our experiments show that the relying party side of the protocol can be implemented with around 6 KBs of code.

A drawback of APCR-LPM is that it requires distribution of symmetric keys prior to protocol run. In the the extended version of our paper (Moustafa et al., 2023), we describe a variant of the protocol where the shared key is generated by the verifier and distributed to RP and attester during the protocol run.

An important feature of our approach is the separation of roles between the relying party and verifier, which delegates the processing of attestation evidence to a third device, critically decreasing the power and memory requirements on the relying party device. The relying party only needs to process attestation result, the verifier's simple, standard-format verdict on the evidence.

REFERENCES

3GPP (2023). Security architecture and procedures for 5G system. Technical Specification TS 33.501 V18.2.0, 3GPP.

Aldoseri, A., Clothia, T., Moreira, J., and Oswald, D. (2023). Symbolic modelling of remote attestation protocols for device and app integrity on Android. In *Proceedings of the 2023 ACM on Asia Conference on Computer and Communication Security*, ASIA CCS'23, pages 218–231, New York, NY, USA. Association for Computer Machinery.

Ammar, M., Crispo, B., and Tsudik, G. (2020). SIMPLE: A remote attestation approach for resource-constrained IoT devices. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*, pages 247–258.

Ammar, M., Washha, M., Ramabhadran, G. S., and Crispto, B. (2018). SlimIOT: Scalable lightweight attestation protocol for the internet of things. In *2018 IEEE Conference on Dependable and Secure Computing (DSC)*. IEEE.

Birkholz, H., Thaler, D., Richardson, M., Smith, N., and Pan, W. (2023). Remote attestation procedures (RATS) architecture. RFC 9334.

Blanchet, B., Smyth, B., Cheval, V., and Sylvestre, M. (2018). *ProVerif 2.00: automatic cryptographic protocol verifier, user manual and tutorial*.

Bormann, C., Ersue, M., and Keranen, A. (2014). Terminology for constrained-node networks. RFC 7228.

Bormann, C. and Hoffman, P. (2020). Concise binary object representation (CBOR). RFC 8949.

Coker, G., Guttman, J., Loscocco, P., Herzog, A., Millen, J., O'Hanlon, B., Ramsdell, H., Segall, A., Sheehy, J., and Sniffen, B. (2011). Principles of remote attestation. *International Journal of Information Security*, 10:63–81.

Dhar, A., Puddu, I., Kostiainen, K., and Capkun, S. (2020). ProximiTEE: Hardened SGX attestation by proximity verification. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, CODASPY '20, page 5–16, New York, NY, USA. Association for Computing Machinery.

Dolev, D. and Yao, A. C. (1983). On the security of public key protocols. *IEEE Transactions on Information Theory*, 29:198–208.

Feng, W., Qin, Y., Zhao, S., and Feng, D. (2018). AAoT: Lightweight attestation and authentication for low-resource things in IoT and CPS. *Computer Networks*, 134:167–182.

Fossati, T., Voit, E., and Trofimov, S. (2023). EAT Attestation Results. https://www.ietf.org/archive/id/draft-fv-rats-ear-01.html. Last Accessed: 17-07-2023.

Gunn, L., Asokan, N., Ekberg, J.-E., Liljestrand, H., Nayani, V., and Nyman, T. (2022). Hardware platform security for mobile devices. *Foundations and Trends in Privacy and Security*, 3:214–394.

Hristozov, S., Wettermann, M., and Huber, M. (2022). A TOUCTOU attack on DICE attestation. In *CODASPY'22: Proceedings of the Twelft ACM Conference on Data and Application Security and Privacy*, pages 226–235, New York, NY, USA. Association for Computing Machinery.

Intel (2017). TinyCrypt Cryptographic Library. https://github.com/intel/tinycrypt.

Jäger, L., Petri, R., and Fuchs, A. (2017). Rolling DICE: Lightweight remote attestation for COTS IOT hardware. In *ARES'17: Proceedings of the 12th International Conference on Availability, Reliability and Security*, ARES '17, New York, NY, USA. Association for Computing Machinery.

Johnson, W. A., Ghafoor, S., and Prowell, S. (2021). A taxonomy and review of remote attestation schemes in embedded systems. *IEEE Access*, 9:142390–14210.

Kohnhäuser, F., Büscher, N., Gabmeyer, S., and Katzenbeisser, S. (2017). SCAPI: A scalable attestation protocol to detect software and physical attacks. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 75–86.

Moustafa, M., Niemi, A., Ginzboorg, P., and Ekberg, J. E. (2023). Attestation with constrained relying party. *arXiv preprint*, abs/2312.08903.

Niemi, A., Bop, V. A. B., and Ekberg, J.-E. (2021). Trusted Sockets Layer: A TLS 1.3 based trusted channel protocol. In Tuveri, N., editor, *Secure IT Systems: 26th Nordic Conference, NordSec 2021*, Lecture Notes in Computer Science, pages 175–191, Cham. Springer International Publishing.

Niemi, A., Nayani, V., Moustafa, M., and Ekberg, J. E. (2023). Platform attestation in consumer devices. In *2023 33rd Conference of Open Innovations Association (FRUCT)*, pages 198–209. IEEE.

Nordic Semiconductor (2023). nRF5340 DK. https://www.nordicsemi.com/Products/Development-hardware/nRF5340-DK. Last Accessed: 17-07-2023.

Parno, B. (2008). Bootstrapping trust in a "trusted" platform. In *Proceedings of the 3rd Conference on Hot Topics in Security*, HOTSEC'08, USA. USENIX Association.

TCG (2021). *DICE Attestation Architecture*. Trusted Computing Group. Version 1.0, revision 0.23.

Zephyr (2023). Bluetooth: IPSP Sample. https://docs.zephyrproject.org/latest/samples/bluetooth/ipsp/README.html. Last Accessed: 17-07-2023.