# Attestation with Constrained Relying Party

**Moustafa., Mariam; Niemi., Arto; Ginzboorg., Philip; Ekberg., Jan-Erik**

*Published in:*
Proceedings of the 10th International Conference on Information Systems Security and Privacy - ICISSP

*Link to article, DOI:*
10.5220/0012319300003648

*Publication date:*
2024

*Document Version*
Peer reviewed version

Link back to DTU Orbit

*Citation (APA):*
Moustafa., M., Niemi., A., Ginzboorg., P., & Ekberg., J-E. (2024). Attestation with Constrained Relying Party. In *Proceedings of the 10th International Conference on Information Systems Security and Privacy - ICISSP* (Vol. 1, pp. 701-708). SCITEPRESS Digital Library. https://doi.org/10.5220/0012319300003648

# Attestation with Constrained Relying Party

Mariam Moustafa[2,3], Arto Niemi[1], Philip Ginzboorg [1] 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 receive privacy-sensitive sensor readings, or to operate a safety-critical actuator, carries significant risk. Usually, such risks are mitigated by validating the device's security state with remote attestation, but current remote attestation protocols are not suitable when the beneficiary of attestation, the relying party, is a constrained device such as a small sensor or actuator. These devices typically lack the power and memory to operate public-key cryptography needed by such protocols, and may only be able to communicate with devices in their physical proximity, such as with the controller whose security state they wish to evaluate. 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 show that our protocol, including the needed cryptography and message processing, can be implemented with a code size of 6 KB and validate its security via model checking with the ProVerif tool.

## 1 INTRODUCTION

Remote attestation allows a device to validate the security state of another device (Coker et al., 2011). A trustworthy mechanism, usually with hardware backing, measures the security state of the device where it resides and generates cryptographically protected attestation evidence. The evidence constitutes a security proof that can be remotely appraised by a verification service, typically by matching measurements reported in the evidence against trusted reference values. The result is a verdict that a relying party can then use as the basis of a trust decision. Securely transmitting these messages – 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. This is usually accomplished with public-key cryptography, such as asymmetric signatures. Critical is also the protocol's resistance against relay attacks, where an attacker uses evidence generated by a valid device to attest a compromised device.

After around two decades of research, remote attestation is now 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 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. Research on resource-constrained attestation focuses on attesting the constrained device: protocols based on PUFs (Sadeghi et al., 2011), one-time signatures (Roberto et al., 2023) and even statistics (Neureither et al., 2020) have been proposed for this use case.

We consider a similar setup but with a reverse attestation requirement: 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 these kinds of settings, the constrained device is a relying party for attestation. A visualization of this is shown in Fig. 1.

As the definition of "constrained", 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. Such a net-

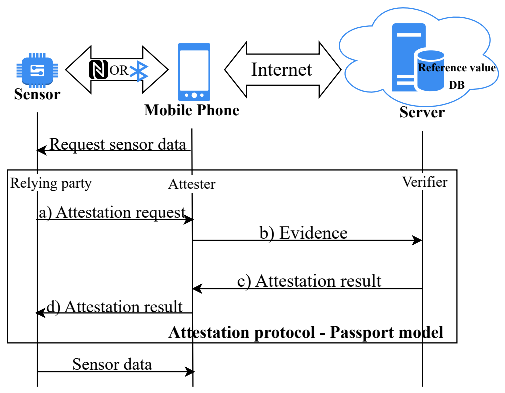


Figure 1: Example of a attestation with a constrained relying party: a sensor validating the security of a mobile phone as a precondition for transmitting sensor readings. The example uses the passport model for communication in a remote attestation, where no direct communication between RP and verifier is needed.

work setup is dominant today — in home IoT Zigbee / BT-LE systems all radio communication is orchestrated via gateways, which in such systems are also the obvious targets of attestation. In personal area networks (PANs) the same applies, and a mobile phone takes the place of the gateway. Examples of PAN devices are phone headsets, heart monitors and blood-sugar sensors.

The IETF RFC 7228 standard (Bormann et al., 2014) categorizes constrained devices into classes based on the amount of memory they have for data and code. The smallest "sensor" Class 0 devices have less than 10 KB data and less than 100 KB code; as asymmetric cryptography implementation typically requires several tens of kilobytes of code, there is no space for it, even if performance and battery consumption concerns are ignored.

We present to our knowledge the first remote attestation protocol that neither uses public-key cryptography in the relying party, nor requires a second (trusted) channel for authentication/attestation. Our contributions are as follows:

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.

## 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)[1], or a Trusted Platform Module (TPM) (Segall, 2017), 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). The RP and verifier roles may be combined and implemented on the same device – this has been common practice in academic work on attestation protocols (Niemi et al., 2022). In the industry, complexity of evidence appraisal – which involves acquiring trusted reference values and matching them against the measurements reported in the evidence – has triggered a shift towards implementing the verifier as a separate online service. This is the more useful setup in the context of constrained devices: separating the two roles reduces the required complexity and code size in the RP device.

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

We will next describe how previous remote attestation protocols cater to constrained devices, focusing on protocols based on symmetric-key cryptography.

The SlimIoT protocol (Ammar et al., 2018) performs swarm attestation where the entities being at-

[1]A TEE is an isolated computing environment in a device. It contains trusted applications that process and store data independently from the device's main operating system.

tested are a ‘swarm’ of constrained IoT devices. The verifier broadcasts two challenges in a sequential manner which are stored by the attesters. Then the verifier discloses the keys used to generate the challenges so that the attesters can verify the stored challenges. If the keys are verified, the attesters generate the attestation evidence, aggregates the evidence from other attesters, and forwards the evidence.

SCAPI (Kohnhäuser et al., 2017) is another swarm-based attestation protocol. It assumes that the attesters contain a Trusted Execution Environment (TEE) to execute tamper-resistant tasks. The verifier issues an attestation request to a specific device which in turn generates an attestation report and requests reports from other attesters in the network. Upon receiving the reports, the attester merges them and sends them to the verifier. 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. The protocol essentially attests key possession and does not consider attestation metrics where additional claims about the device are included.

AAoT (Feng et al., 2018) is an attestation protocol based on physical unclonable functions (PUFs) – special hardware that is used to generate the symmetric keys between the attester and verifier. The first stage of the protocol includes mutual authentication where the verifier and attester generate keys based on their identities using PUFs and performing a MAC based on their previously exchanged messages. If the MAC digests do not match, the protocol is aborted. Otherwise, the attester checksums the entire memory content in the device including the PUF component.

In the SIMPLE protocol (Ammar et al., 2020), the verifier generates a nonce, the value of a valid software state, and the MAC of these values. The attester verifies the MAC, computes its own state, and checks whether the computed state matches that the verifier sent. The attester sends the results of its check to the verifier to validate.

All in all, these protocols cater to the case where the attester, instead of the relying party (RP), is the constrained device. This is evidenced by the lack of separation between verifier and RP roles. When the RP and verifier roles are both implemented on the same constrained device, only simple attestation metrics and appraisal procedures are possible, since verification is constrained by the resource limits of the RP device. Another drawback of some of the protocols is that they require synchronized clocks, or special hardware, like PUFs, in the constrained device.

## 3 REQUIREMENTS

We list below the functional and security requirements for the remote attestation protocol with a constrained relying party.

### 3.1 Functional Requirements

To ensure the viability of our protocol in the use case where the relying party (RP) is a constrained device, such as a class 0 or class 1 sensor or actuator, 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 and Table 1 describes our notation. 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, (often denoted a "Trusted Execution Environment" (TEE)).
- Verifier V: a non-constrained device, such as a cloud server, that is trusted by both RP and attester. The verifier can validate and appraise the evidence sent by the attester. RP and verifier are assumed to have agreed upon an evidence appraisal policy.

**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. We give a variant of our protocol for this scenario in the Appendix.

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).

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.

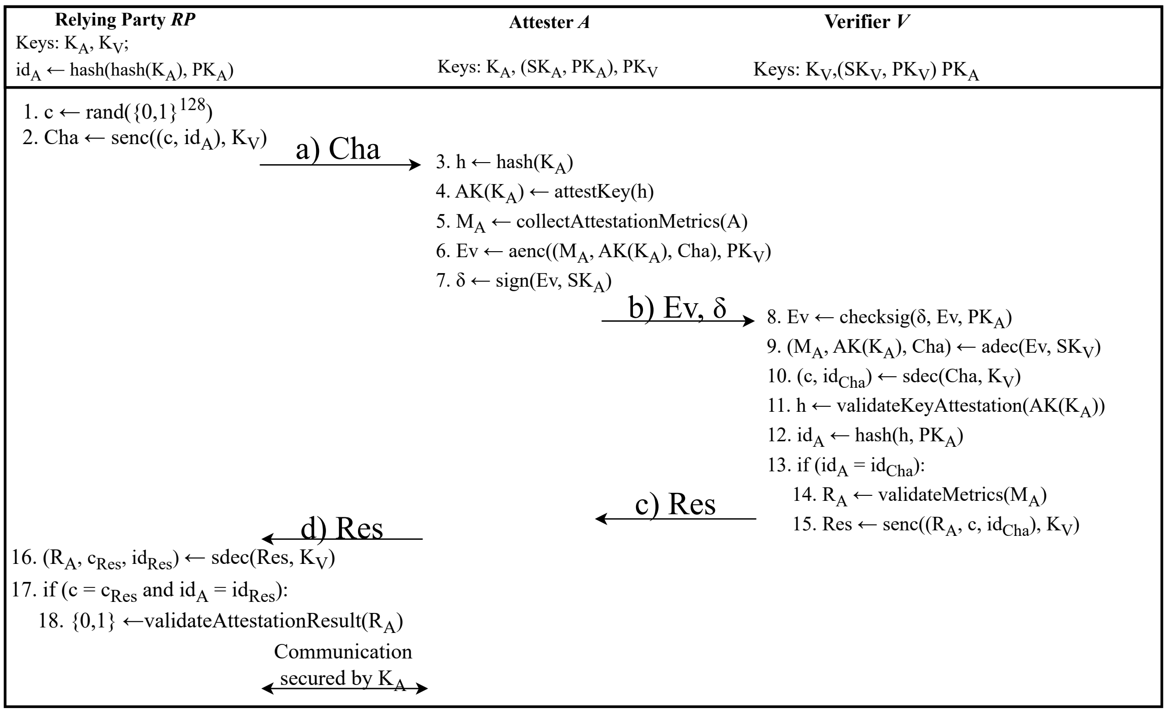

Relying Party *RP*
Keys: $K_A$, $K_V$;
$id_A \leftarrow hash(hash(K_A), PK_A)$
Attester *A*
Keys: $K_A$, $(SK_A, PK_A)$, $PK_V$
Verifier *V*
Keys: $K_V$,$(SK_V, PK_V)$ $PK_A$
1. $c \leftarrow rand(\{0,1\}^{128})$
2. $Cha \leftarrow senc((c, id_A), K_V)$
a) Cha
3. $h \leftarrow hash(K_A)$
4. $AK(K_A)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

Figure 2: Attestation Protocol for Constrained Relying Party.

(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

We analyze the security of ACPR-LPM first via informal discussion and then formal model checking.

### 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

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 bits 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 the metrics $M$. |
| $\{0,1\} \leftarrow validateAttestationResult(R)$ | Determine trustworthiness based on the attestation results $R$. |

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 [2].

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*);

[2]ProVerif Model

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. The $M_A$ is matched to both the attester and verifier event 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.

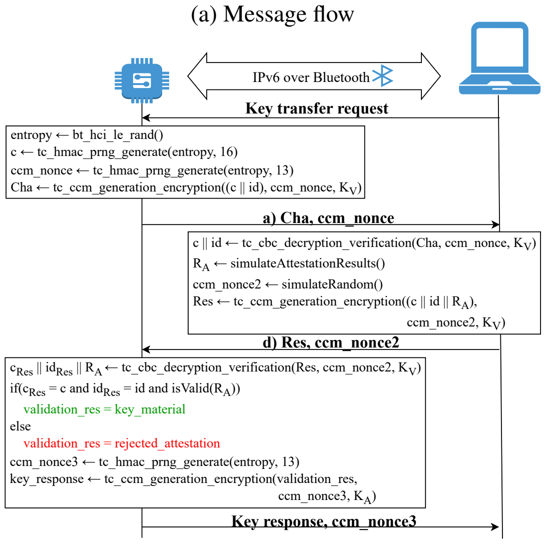
Our attestation protocol is applicable to this setup as follows. The keytag (relying party) initiates the attestation protocol over a near-field communication channel with the smartphone (attester), who in turn relies on a network server (verifier) to prove the security level of the TEE in the smartphone. After successful verification of attestation results, the keytag transfers the door key to the smartphone TEE.

### 6.1 Board Setup

The nRF5340 board has a 64 MHz Arm Cortex-M33 CPU, 256 KB Flash and 64 KB RAM. It supports many interfaces including Bluetooth Low Energy and NFC. The manufacturer provides an SDK that includes the Zephyr RTOS (Real-Time Operating System), an operating system based designed for resource constrained devices. Zephyr can run on devices with 32KBs of RAM and has built-in support for multiple cryptographic libraries. For cryptography, we used primitives from the TinyCrypt library (Intel, 2017), namely AES-CCM, PRNG and HMAC.

We extended Zephyr's sample application, called IPSP (Zephyr, 2023), with the implementation of the key transfer protocol shown in Fig. 3a. The connection between the relying party and attester in this prototype is IPv6 over BLE (Bluetooth Low Energy).

(a) Message flow



```
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, $K_A$)
```

(b) EAR object

```
{
    "eat_profile": "tag:github.com,2023:veraison/ear",
    "iat": 1666529300,
    "ear.verifier-id": {
        "developer": "https://veraison-project.org",
        "build": "vts 0.0.1"
    },
    "submods": {
        "Mobile Phone": {
            "ear.status": "affirming"
        }
    }
}
```

Figure 3: Prototype implementation of keytag application.

## 6.2 Implementation

As illustrated in Fig. 3a, 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 the values $c$ and $id$. Then it encodes the attestation result object and sends the encrypted $c$, $id$, and $R_A$ over the Bluetooth channel. The attestation result object follows the Entity attestation token Attestation Result (EAR) (Fossati et al., 2023) emerging standard. An example of an EAR JSON object is shown in Figure 3b. The displayed required fields contain information about the EAR version, the issued at time, the verifier identity, the attester's identity and the verifier's verdict. The result object is encoded using the CBOR binary serialization format (Bormann and Hoffman, 2020) as CBOR has a small code footprint and encoding 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. The current prototype does not implement all the steps in the attestation protocol; it is mainly concerned with implementing the relying party application.

## 6.3 Measurements

We compared the RP application against the original IPSP application to determine the increase in RAM and Flash requirements, shown in Table 2. The Flash memory increased by 6 KB and the RAM increased by less than 1 KB. This includes protocol processing, code from the TinyCrypt and zcbor libraries, and code specific to the keytag application. To exclude Zephyr from the measurements, we analyzed the object files and the size of the program was computed. The RP implementation, including protocol processing and code that calls TinyCrypt or zcbor, required around 2.3 KB of extra code. The total flash and memory size are the reported values Zephyr displays when building the application. The application code size was determined using the `size` command-line tool on the RP application's object file.

The Zephyr ELF (Executable and Linkable Format) file was analyzed to determine the extra space the TinyCrypt and zcbor library have used. The command "nm –size-sort –radix=d zephyr.elf" was used to extract the sizes of the library functions used in the relying party application. Only the text size was measured as it is the application code being executed. Table 3 shows the results. The application object file was analyzed in the same manner to determine the size of the different components of the code. The results are shown in Table 4.

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) |

Table 3: Libraries flash code size.

| Libraries Flash | Size [bytes] |
|---|---|
| TinyCrypt | 2304 |
| zcbor | 398 |

Table 4: Application components code size.

| Application code | Size [bytes] |
|---|---|
| Initialization & Logging | 238 |
| Network Stack | 1606 |
| Cryptography | 272 |
| EAR decoder | 542 |

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 RESULTS AND DISCUSSION

The APCR-LPM does not need asymmetric cryptography in the relying party, making it suitable for devices with limited resources. The measurements summarized in Section 6.3 show that the protocol is sufficiently small to be embedded in simple devices. According to the formal verification by ProVerif, the protocol is secure under an active Dolev-Yao attacker.

Previous remote attestation protocols for constrained devices, discussed in Section 2.2, either combine the relying party and verifier roles in the same device, with significant drawbacks in memory footprint and complexity, or require certain hardware in the constrained devices for the protocol to work. In practice, a relying party might be a network gateway, actuator, sensor, or keytag that may not be able to perform all the functionality that is requested of a verifier. Also, a protocol that does not require special hardware is more scalable. Table 5 analyzes the differences between the earlier protocols and APCR-LPM. The separation of roles between the relying party and verifier in our protocol allows processing complex attestation evidence without increase in the memory footprint or processing power in the relying party. The relying party only needs to process attestation result, a standardized verifier's verdict on the evidence, following, e.g., the EAR proposal (Fossati et al., 2023).

A drawback of APCR-LPM is that it requires distribution of symmetric keys prior to protocol run. How to best achieve this depends on the relation between the parties. In the Appendix we describe a variant of the protocol that does not involve prior key distribution between the relying party and attester: the shared key is generated by the verifier and distributed to RP and attester during the protocol run. The flip side of this is that the verifier knows the key shared between the RP and the attester.

## 8 CONCLUSION

We have designed and implemented APCR-LPM – an attestation protocol for situations where the beneficiary of attestation, the relying party, is a very simple device, capable only of symmetric key operations and able to communicate only with the device whose trustworthiness it wants to evaluate. Our example use case is an electronic lock-and-key system, where the device generating the key is a dongle. In our imple-

Table 5: Differences between APCR-LPM and other symmetric encryption-based RA protocols. The meaning of the columns is as follows: "Separate RP": the RP and verifier roles are separated. "HW-independence": the protocol can be implemented on any hardware. "HW-based RA": attestation uses a hardware-backed trustworthy mechanism. "Universal attestation": the protocol can be used to carry any attestation evidence format and arbitrary attestation claims instead of a small or fixed set.

| **Protocol** | **Separate RP** | **HW-independence** | **HW-based RA** | **Universal attestation** |
|---|---|---|---|---|
| SlimIoT (Ammar et al., 2018) | ✗ | ✗ | ✓ | ✓ |
| SCAPI (Kohnhäuser et al., 2017) | ✗ | ✗ | ✓ | ✓ |
| Rolling DICE (Jäger et al., 2017) | * | ✓ | ✓ | ✗ |
| AAoT (Feng et al., 2018) | ✗ | ✗ | ✓ | ✗ |
| SIMPLE (Ammar et al., 2020) | ✗ | ✓ | ✗ | ✗ |
| APCR-LPM [this paper] | ✓ | ✓ | ✓ | ✓ |

(*) No RP but theoretically the roles of attester and verifier can be reversed so that verifier is a constrained device.

mentation, the constrained device is an nRF5340 development board which connects via Bluetooth to a Linux laptop and performs relying party role in our protocol. The application requires 6 KB of code and 287 ms to perform the key transfer and attestation functionality. We used ProVerif to verify the protocol's security properties. In conclusion, the protocol is practical for the intended use cases.

## 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. *Version from*, pages 05–16.

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 *Pro-*

*ceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 75–86.

Neureither, J., Dmitrienko, A., Brasser, F., and Sadeghi, A.-R. (2020). LegIoT: Ledgered trust management platform for IoT. In *Computer Security – ESORICS 2020*, pages 377–396, Cham. Springer International Publishing.

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.

Niemi, A., Sovio, S., and Ekberg, J.-E. (2022). Towards interoperable enclave attestation: Learnings from decades of academic work. In *2022 31st Conference of Open Innovations Association (FRUCT)*, pages 189–200. 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.

Rescorla, E. (2018). The Transport Layer Security (TLS) Protocol version 1.3. RFC 8446.

Roberto, R., Arjona, R., and Baturone, I. (2023). A lightweight remote attestation using PUFs and hash-based signatures for low-end IoT devices. *Future Generation Computer Systems*, 148:425–435.

Sadeghi, A.-R., Schulz, S., and Wachsmann, C. (2011). Lightweight remote attestation using physical functions. In *WiSec'11: Proceedings of the fourth ACM conference on Wireless network security*, pages 109–114. ACM.

Segall, A. (2017). *Trusted Platform Modules: Why, when and how to use them*. Institution of Engineering and Technology, London, United Kingdom.

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.

## APPENDIX

We will now describe a variant of the protocol where the distribution of shared key to RP and attester is done during the protocol run.

Initially, the relying party (RP) has a symmetric key $K_V$ that is unique to the RP device and shared with the verifier (V). The attester has two asymmetric keypairs $(SK_A, PK_A)$ and $(SK'_A, PK'_A)$ for signing and encryption, respectively. The hash of $PK_A$ is denoted by $h$. The verifier has two asymmetric keypairs $(SK_V, PK_V)$ and $(SK'_V, PK'_V)$, for signing and encryption, respectively. The verifier and the attester know each other's public keys.

After the protocol run the RP has validated the result of the verifier's assessment of the attester's state; and the RP and the attester have a verifier-generated session key $K_S$.

The protocol steps are as follows (cf. Figure 4). The attester sends to the RP message (a) containing the hash $h$ of its public key $PK_A$. (1) The RP prepares the challenge it will send to the attester by generating the nonce $c$. (2) It creates the challenge Cha by encrypting using $K_V$ the nonce $c$ and $h$, and sends Cha to the attester in message (b). The attester cannot decrypt Cha, because it does not have $K_V$.

(3) The attester collects the attestation metrics $M_A$ of the device, and (4) creates the evidence Ev, comprising of $M_A$, $h$, and Cha encrypted with $PK'_V$. (5) It signs the Ev with its private key $SK_A$, obtaining the signature $\delta_A$, and sends the Ev and $\delta_A$ to the verifier in message (c).

(6) Upon receiving message (c), the verifier checks the signature $\delta_A$. If this check succeeds, it decrypts the Ev using the key $SK'_V$, and obtains $M_A$, $h_A$, and Cha; otherwise, the verifier aborts the protocol. (8) It decrypts the challenge Cha to extract $c$ and $h_{RP}$. (9) If $h = h_A$ and $h_A = \text{hash}(P_A)$, then the protocol run continues as follows.

(10) Based on the metrics $M_A$, the verifier creates the attestation result $R_A$; (11) generates a session key $K_S$; and (12) encrypts $R_A$, $c$, $h_{\text{RP}}$, $K_S$ with the key $K_V$ it shares with the RP, resulting in $Res_{\text{RP}}$. Next, (13) the verifier encrypts $Res_{\text{RP}}$ and $K_S$ using the public key $PK_A$ of the attester, and (14) signs the resulting $Res_A$ using his secret key $SK_V$. The verifier sends $Res_A$ and the signature $\delta_V$ in message (d).

(15) Upon receiving message (d), the attester checks the signature $\delta_V$. (16) If this check succeeds, it decrypts $Res_A$ using its secret key $SK_A$, and obtains $Res_A$ and $K_S$ from $Res_A$; otherwise, the attester aborts the protocol. The attester sends $Res_{\text{RP}}$ to the RP in message (e).

(17) Upon receiving message (e), the RP decrypts $Res_{\text{RP}}$ and obtains $R_A$, $c_{\text{Res}}$, $h_{\text{Res}}$ and $K_S$. (18) If $c = c_{\text{Res}}$ and $h = h_{\text{Res}}$, then (19 ) RP will process the attestation result to determine the attester's state; otherwise, it will abort the protocol run.

Subsequent, application-specific communication between RP and attester (f) depends on the result of step (19); this communication is secured using

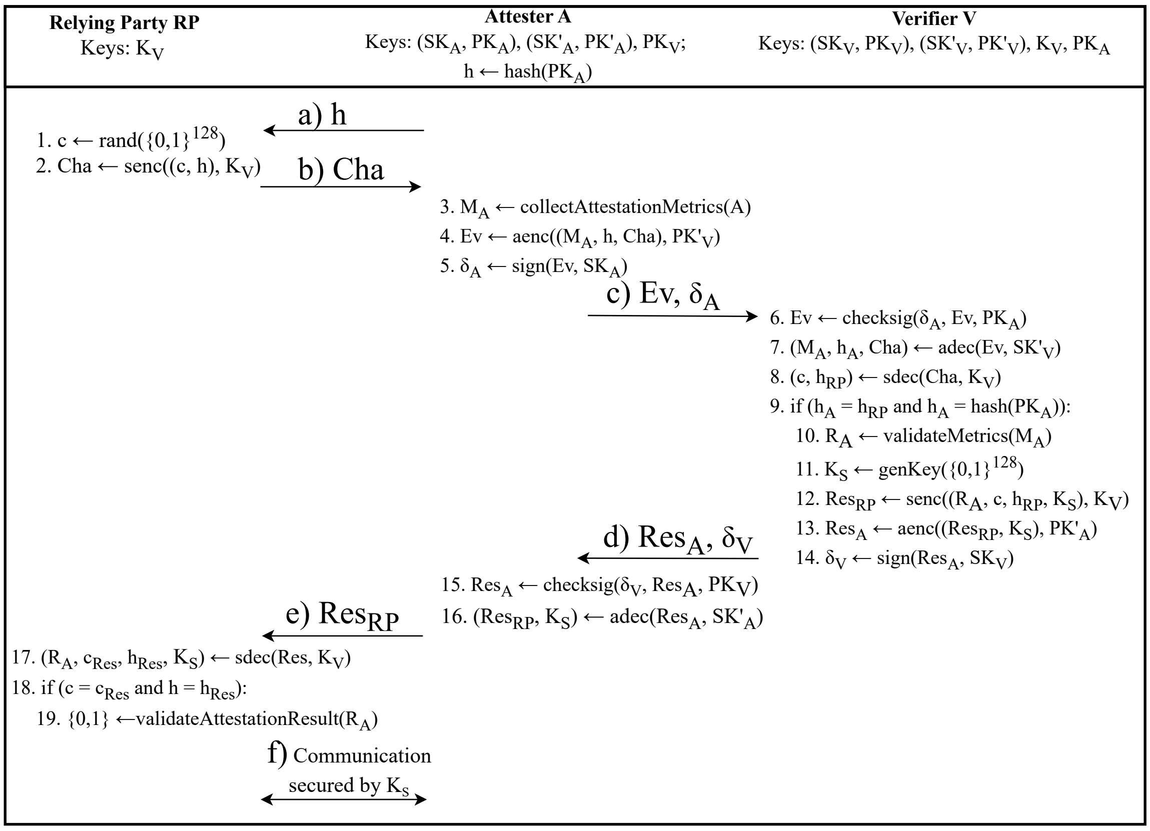

Relying Party RP
Keys: $K_V$
Attester A
Keys: $(SK_A, PK_A)$, $(SK'_A, PK'_A)$, $PK_V$;
$h \leftarrow hash(PK_A)$
Verifier V
Keys: $(SK_V, PK_V)$, $(SK'_V, PK'_V)$, $K_V$, $PK_A$
a) h
1. $c \leftarrow rand(\{0,1\}^{128})$
2. $Cha \leftarrow senc((c, h), K_V)$
b) Cha
3. $M_A \leftarrow collectAttestationMetrics(A)$
4. $Ev \leftarrow aenc((M_A, h, Cha), PK'_V)$
5. $\delta_A \leftarrow sign(Ev, SK_A)$
c) $Ev, \delta_A$
6. $Ev \leftarrow checksig(\delta_A, Ev, PK_A)$
7. $(M_A, h_A, Cha) \leftarrow adec(Ev, SK'_V)$
8. $(c, h_{RP}) \leftarrow sdec(Cha, K_V)$
9. if $(h_A = h_{RP}$ and $h_A = hash(PK_A))$:
10. $R_A \leftarrow validateMetrics(M_A)$
11. $K_S \leftarrow genKey(\{0,1\}^{128})$
12. $Res_{RP} \leftarrow senc((R_A, c, h_{RP}, K_S), K_V)$
13. $Res_A \leftarrow aenc((Res_{RP}, K_S), PK'_A)$
14. $\delta_V \leftarrow sign(Res_A, SK_V)$
d) $Res_A, \delta_V$
15. $Res_A \leftarrow checksig(\delta_V, Res_A, PK_V)$
16. $(Res_{RP}, K_S) \leftarrow adec(Res_A, SK'_A)$
e) $Res_{RP}$
17. $(R_A, c_{Res}, h_{Res}, K_S) \leftarrow sdec(Res, K_V)$
18. if $(c = c_{Res}$ and $h = h_{Res})$:
19. $\{0,1\} \leftarrow validateAttestationResult(R_A)$
f) Communication
secured by $K_S$

Figure 4: Variation of APCR-LPM with no pre-shared keys between relying party and attester.

$K_S$. Please note that before that communication takes place, the RP and attester have not confirmed the possession of $K_S$ to each other.

The requirements for this protocol are as in Section 3, with the addition of SR5: secrecy of $K_S$. We have verified the security properties SR1-SR5 using ProVerif. Our code is available in GitHub [3].

We remark, first, that an alternative to verifier generating the session key $K_S$ (step (11) in Figure 4), is to generate $K_S$ in the RP, and then send it encrypted with $K_V$ to the verifier in message (a). Second, the protocol description could be simpler if rather than separately protecting messages (c) and (d), the attester and the verifier would communicate through a secure channel using, e.g., the TLS 1.3 protocol (Rescorla, 2018).

[3] ProVerif Model