Establishing Secure Communication Channels Using Remote Attestation with TPM 2.0

Paul Georg Wagner[1(✉)], Pascal Birnstill[2], and Jürgen Beyerer[1,2]

[1] Karlsruhe Institute of Technology, Karlsruhe, Germany
paul.wagner@kit.edu
[2] Fraunhofer Institute of Optronics, System Technologies and Image Exploitation IOSB, Karlsruhe, Germany

Abstract. Remote attestation allows a verifier to remotely check the integrity of a trusted computing platform. In recent years a number of attestation protocols based on Trusted Platform Modules (TPMs) have been proposed. These protocols cryptographically verify a trusted platform's state by exchanging TPM-signed quotes. Some of them also establish an encrypted channel to the trusted platform, which allows the verifier to transmit information that only the attested software stack can read. However, many existing attestation protocols are either vulnerable against man-in-the-middle attacks, or depend on outdated TPM specifications. In this work we analyze a recently developed attestation protocol that is being actively used to interconnect highly distributed trusted applications. We find this protocol to be vulnerable against a variant of the classical relay attack. In response to this threat we develop a lightweight remote attestation protocol based on the TPM 2.0 specification that is not vulnerable to this attack. Unlike previous proposals, our protocol relies solely on the TPM to establish a shared key on the attested channel, which significantly reduces its attack surface. Our protocol supports mutual attestation, perfect forward secrecy and is independent of the underlying network stack. We provide a reference implementation of our protocol and compare its performance to previous proposals. We also analyze its security properties using the Tamarin theorem prover.

Keywords: Trusted computing · Trusted Platform Modules · Remote attestation · Key establishment · Secure channels · Attestation protocols

1 Introduction

Trusted Platform Modules (TPMs) are tamper-resistant hardware chips that extend computer systems with basic security related features. Similar in nature to smart cards, TPMs provide a hardware implementation of cryptographic functions and protected storage for cryptographic keys. Since they are included in many motherboards, TPMs are arguably the most prevalent trusted computing

K. Markantonakis and M. Petrocchi (Eds.): STM 2020, LNCS 12386, pp. 73–89, 2020.
https://doi.org/10.1007/978-3-030-59817-4_5

technology today. One of the most interesting TPM capabilities is the remote verification of a trusted platform's software stack. For this the TPM uses volatile platform configuration registers (PCRs) to measure the current hardware and software configuration as an unforgeable fingerprint. A remote verifier can then request proof of the platform configuration in form of a quote that has been signed by the TPM. This quote contains a nonce generated by the verifier as well as the trusted platform's fingerprint, thereby attesting to the trusted software stack. After comparing the fingerprint to expected values, the verifier is convinced that the trusted platform indeed runs a legitimate software stack. Remote attestation protocols define how the verification of a trusted platform should be conducted over the network. Besides creating and transmitting TPM-signed quotes, most protocols also establish encrypted channels between the attested endpoints. Even though this has been a topic of comprehensive research in the past, currently there are very few ready-to-use protocol implementations available. One of the few actively used remote attestation protocols is the *Industrial Data Space Communication Protocol (IDSCP)* [10]. Introduced by Lux and Brost in 2018, IDSCP has been developed specifically for the Fraunhofer Industrial Data Space (IDS) project [12], which provides secure virtual data spaces for smart business ecosystems. The IDSCP protocol incorporates the benefits of several earlier proposals, such as TLS integration as well as support for mutual attestation and perfect forward secrecy. Also its code is publicly available on Github[1]. Because of these advantages, IDSCP is currently one of the most advanced attestation protocol in productive use.

In this work we present an attack on the IDSCP protocol in scenarios where multiple software components are providing an attestation endpoint on the trusted platform. This scenario is especially relevant in applications where we cannot predict what software will be considered trustworthy, but it applies to other use cases as well. In response to this threat we also develop a lightweight remote attestation protocol based on TPM 2.0 that is not vulnerable to this attack, while still keeping the advantages of IDSCP over previous proposals. Our protocol is easy to implement, supports mutual attestation and forward secrecy, and is independent of the used network stack. In addition our protocol keeps the secret parts of the shared keys used to encrypt the attested channel inside the TPM at all times, which is a security advantage over previous proposals. A reference implementation of our protocol as well as the formal model used for security verification are publicly available[2]. The remainder of this paper is structured as follows. In Sect. 2 we discuss the strengths and weaknesses of existing remote attestation protocols, before briefly presenting IDSCP and the vulnerabilities we found with it. In Sect. 3 we describe the design of our protocol and its reference implementation. Finally, in Sect. 4 we evaluate the proposed protocol in terms of functionality and security. We also verify the resilience of our protocol against the presented attack using the Tamarin theorem prover [11].

[1] https://github.com/industrial-data-space.
[2] https://gitlab-ext.iosb.fraunhofer.de/wagner/tpm20-attestation-protocol.

2 Discussion of Existing Protocols

In this section we first define requirements that attestation protocols should adhere to. Then we discuss the most important remote attestation protocols that have been proposed so far and point out their strengths and weaknesses. Finally we present a still unpublished attack on IDSCP.

2.1 Protocol Requirements

We define three *security requirements* S1 to S3 for remote attestation protocols.

(S1) Authentication: The protocol verifier unambiguously identifies and authenticates the attested platform. No attacker may forge the authentication.
(S2) Platform integrity verification: The protocol verifier is convinced that the attested platform runs a certain trusted software stack, i.e. its PCRs are set to specific values. No attacker is able to impersonate a trusted platform.
(S3) Secure key establishment: The protocol verifier establishes a shared secret with the attested platform. No attacker is able to determine this secret.

Any successful attack against the three main security requirements breaches the confidentiality and integrity of information transmitted over the secure channel. In order to represent security properties of the discussed protocols in finer detail, we distinguish four types of attacks with the additional requirements S4 to S7. Our attacker model consists of adversaries who control the network as well as any secret outside the TPM, including platform secrets such as TLS private keys.

(S4) No replay attacks: No attacker is able to break S1 to S3 by resending previously intercepted legitimate messages.
(S5) No insider attacks: No attacker with knowledge of platform secrets that are not protected by the TPM (e.g. TLS private keys) may break S1 to S3.
(S6) No relay attacks on protocol endpoint: No attacker is able to launch a relay attack targeting the specific protocol's attestation endpoint.
(S7) No relay attacks on non-protocol endpoint: No attacker is able to launch a relay attack targeting any other available attestation endpoint.

In addition to the security goals we define four *functional requirements* F1 to F4. They deal with protocol properties necessary to apply the solution to a wide range of dynamic and challenging tasks.

(F1) Mutual attestation: Establishing a secure channel through a single protocol handshake verifies the platform integrity of both peers.
(F2) Re-attestation: Any verifier can re-attest the trusted platform even after the protocol has established a shared secret.
(F3) Forward secrecy: Disclosing a long-term secret must not corrupt previously intercepted protocol sessions.
(F4) Protocol overhead: The attestation protocol must not generate substantial overhead in performance, implementation complexity and code base size.

2.2 Related Work

In recent years several remote attestation protocols have been proposed. However, most of them are not suitable for large-scale remote attestation applications, or have already been identified as vulnerable.

No Key Establishment or Attestation. Some protocols such as [5,13] only focus on conducting a remote attestation and do not include a shared key establishment (requirement S3). Since these attestation protocols cannot establish secure communication channels, they are unsuitable for scenarios where confidential information should be transmitted to a remote trusted software stack. A more recent project integrates the TPM 2.0 engine in OpenSSL[3]. However, this only allows OpenSSL to use the TPM hardware for its cryptographic operations, but does not provide a remotely attested secure channel (requirements S2 and S3).

Vulnerable Against Insider Attacks. Several protocols provide a secure channel between the verifier and the attested platform by combining remote attestation techniques with an underlying TLS protocol instance. When doing so it is vital to properly bind the keys responsible for creating the encrypted TLS channel to the attested trusted platform, because otherwise the protocols can become insecure. Cheng et al. [4] propose to establish the link between remote attestation and TLS by hashing the TLS pre-master secret into the quote. However, this approach is insecure against attackers with insider knowledge, such as administrators of attested systems. We have to assume that system administrators have access to long-term platform secrets, as long as they are not protected by the TPM. Hence they can simply use the TLS private keys to decrypt the pre-master secret that the legitimate trusted platform transmits during the TLS handshake. As a result, they can passively intercept secrets that an unsuspecting verifier sent across the trusted channel to the attested system. Remote attestation protocols susceptible to this kind of insider attack do not fulfill security requirement S5. Similarly Goldman et al. [7] attempt to link remote attestation with the SSL/TLS stack by adding an intermediate platform certificate signed by a trusted CA. This method also does not protect against insider attacks, because we have to assume that an insider attacker can obtain the unprotected private key corresponding to this new platform certificate. Zhou and Zhang [16] base their solution to this problem on pre-shared passwords. However, this approach is also vulnerable against malicious administrators, because again we have to assume that they know any pre-shared password. Furthermore a protocol based on pre-shared passwords is unfeasible for highly distributed use cases where new communication endpoints regularly join the network (requirement F4).

Vulnerable Against Relay Attacks. Protocols are vulnerable against insider attacks if they use long-term secrets to establish the secure channel. This attack can be avoided by conducting an ephemeral key exchange during the attestation process. Stumpf et al. [14] propose to link remote attestation with a Diffie-Hellman key exchange by including the ephemeral public keys in the quote.

[3] https://github.com/tpm2-software/tpm2-tss-engine.

After the attestation is complete, both endpoints derive a symmetric key from the established secret and use it to encrypt all subsequent communication. While this thwarts the insider attack and provides forward secrecy (S5 and F3), including ephemeral public keys in the quote can introduce vulnerabilities against relay attacks (c.f. [6,16]). During this attack the adversary relays messages to the original trusted system in order to answer another attestation challenge on his own. Protocols directly based on Stumpf's approach [1,8] are also susceptible to this attack and do not fulfill requirement S6.

Other Protocols. Gasmi et al. [6] and Armknecht et al. [2] embed the remote attestation process into a standard TLS handshake. While this fixes the vulnerability against relay attacks, there are drawbacks to this approach. Since attestation data is included in the TLS handshake, a modified TLS implementation has to be used. Tying remote attestation to the TLS library conflates protection goals and complicates code base updates (requirement F4). Furthermore, these protocols use the *Secret Key Attestation Evidence (SKAE)* feature of the TPM 1.2 specification, which is not included in recent versions anymore. In a similar fashion, Lan et al. [9] add direct anonymous attestation (DAA) to a TLS handshake. This approach also uses a modified TLS stack, does not support mutual attestation and requires fresh AIK re-enrollments for each handshake, which is very costly in distributed environments (requirements F1 and F4). Also neither of these proposals offer working code or usable libraries.

We provide a detailed comparison of the discussed protocols during the evaluation in Sect. 4 (c.f. Tables 2 and 3). All in all no existing protocol fulfills all requirements we have for a remote attestation protocol.

2.3 Attacks on IDSCP

The Industrial Data Space Communication Protocol (IDSCP) is a remote attestation protocol introduced by Lux and Brost in 2018 [10]. It relies on an underlying standard TLS connection to provide authentication and channel encryption. However, unlike with previous TLS-based proposals, the remote attestation is not directly included in the TLS handshake. Instead a mutual attestation is conducted as soon as the standard TLS connection has been established between both trusted endpoints. During this attestation phase both sides draw and exchange random nonces to protect against replay attacks. These nonces are then used to generate a TPM-signed quote containing a selection of the current PCR values on each trusted platform. Finally the quotes' PCR values and signatures are verified by each side, thereby mutually confirming the integrity of both trusted software stacks. In order to link the underlying TLS session with the conducted remote attestation, the quotes also contain a hash of the respective TLS certificates that authenticated the initial TLS handshake. By comparing this hash with his own TLS public key fingerprint, each verifier can check if the attested remote endpoint operates on the "right" TLS channel. A complete description of the IDSCP protocol is given in [10]. For convenience purposes we include an overview of the protocol handshake in the appendix (c.f. Table 4).

Instead of conducting a separate key agreement, IDSCP relies on the underlying TLS channel to encrypt transmitted data. Since no modified or enhanced TLS handshake is necessary, IDSCP can be easily implemented using any standard TLS library. This is a clear advantage over previous TLS-based remote attestation protocols such as [4] and [6]. However, conflating the security goals of the encryption layer and the attestation protocol introduces new vulnerabilities. The attestation part of IDSCP simply presupposes the security of the encrypted TLS channel without considering that the respective attacker models differ. As a result, the protocol becomes vulnerable against an insider attack similar to the previously mentioned one on Cheng's proposal [4], if the underlying TLS key establishment is performed by transmitting an encrypted pre-master secret. An insider attacker, who has access to the TLS private keys of the trusted platform, can intercept the pre-master secret and subsequently decrypt any communication that should be decipherable solely by the trusted software stack of the attested endpoint. Even though not explicitly mentioned by the IDSCP specification, this attack can be avoided by forcing both endpoints to agree on a TLS cipher suite that features perfect forward secrecy (PFS).

Scenario for a Relay Attack. However, even when IDSCP uses perfect forward secrecy, in certain situations the protocol is still vulnerable against a variant of the classical relay attack. During a relay attack the adversary typically conducts a legitimate remote attestation with the trusted platform and uses the response to forge attestation evidence for his own platform. While relay attacks on Stumpf's protocol are performed this way (c.f. [6,16]), the attack on IDSCP requires a modified approach. More concretely, we extend the classical relay attack by using a legitimate third-party application on the trusted platform that also offers an attestation endpoint. The key to this attack is that these two endpoints, while on the same platform, do not implement the same attestation protocol. Even though the security impact of multiple attestation endpoints is seldom being considered, this is a very realistic attack vector for IDSCP. Since the Industrial Data Space architecture supports distributed data processing across multiple companies [12], third-party applications are deployed on trusted platforms.

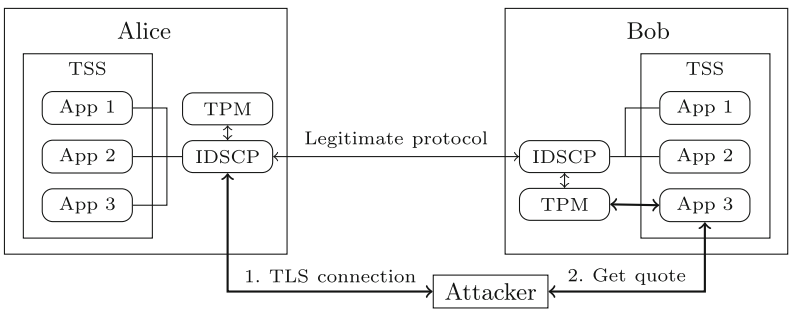

Fig. 1. Scenario for the relay attack on IDSCP.

As a result it is possible that – in addition to the general IDSCP endpoint – a second remote attestation endpoint is provided by one of the legitimate data processing applications. This scenario is depicted in Fig. 1. The communication is routed through the IDSCP endpoints on both Alice's and Bob's system. In order to intercept information intended for Bob's trusted software stack, the attacker first establishes a TLS connection with Alice's IDSCP endpoint. Then the attacker contacts the additional attestation endpoint in Bob's trusted software stack (in this case app 3) to retrieve the quoting information he needs to complete the IDSCP handshake. Note that app 3 is a legitimate data processing application and a valid part of Bob's trusted software stack (TSS), so the PCR fingerprints of the generated quote are as Alice expects them to be. Hence Alice will trust this IDSCP channel, even though it is controlled by the attacker.

Relay Attack on IDSCP. If the IDSCP protocol is executed properly, each trusted platform first performs a TLS handshake, chooses the Diffie-Hellman key pair and signs the public part with the TLS private key. In order to extract information intended only for a trusted platform, the attacker has to perform these steps on a different machine. Fig. 2 shows how an internal attacker can intercept encrypted information from an IDSCP channel. In this example Alice acts as honest verifier, who intends to establish an attested IDSCP connection to Bob's trusted platform. As before, we assume the strongest attacker to be an administrator attempting to intercept information that should be decipherable only by Bob's unmodified trusted software stack. This attacker has access to Bob's TLS long-term secrets and can use them to conduct his own TLS handshake with the remote verifier (Alice), thereby impersonating an endpoint on Bob's legitimate trusted system (1). The goal of the attack is to establish an attested connection with a computer system that is not the trusted platform it claims to be. Note that even though an insider attacker typically has access to Bob's legitimate trusted platform, he cannot use the original system to do this, because that would change the PCRs and hence reveal the attack.

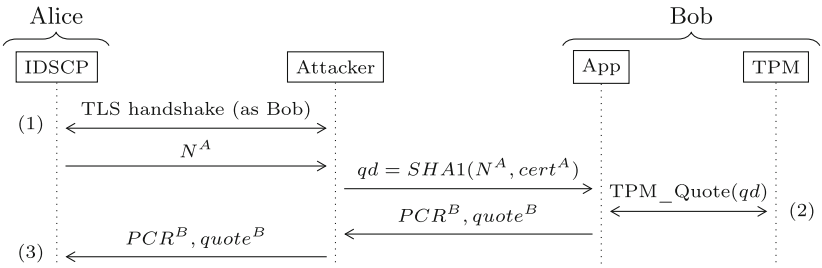

Fig. 2. Relay attack on the IDSCP protocol. Not all messages are shown.

After the TLS connection is established, the attacker needs to present a valid quote during the subsequent attestation phase of the IDSCP protocol. This quote

needs to be signed by Bob's original TPM and has to contain both the correct nonce and the remote TLS certificate. Clearly such a quote can only be created by Bob's original trusted platform. In a classical relay attack the adversary would try to extract the quote by initiating another IDSCP handshake with Bob's trusted platform and relay Alice's nonce as his own. However, this approach does not work for IDSCP, because Alice expects the quote to contain both her nonce and her TLS certificate. According to the protocol specification, Bob's unmodified IDSCP endpoint only generates quotes containing TLS certificates that the remote party used for establishing the connection. Hence the attacker would need to sign a key exchange with Alice's TLS long-term secret, which he does not know. Instead he contacts the additional attestation endpoint in Bob's trusted software stack and requests a quote from there. To make the quote look as if generated by the IDSCP endpoint, the attacker calculates the hash of Alice's nonce and her TLS certificate – both values are known to him – and uses it as his own nonce (2). Assuming that the additional attestation endpoint processes the requested qualifying data unaltered, this generates a quote that convinces Alice (3). Since the hash value is indistinguishable from a random nonce, neither honest party can detect this interference. Once the handshake is completed, Alice is confident that she can securely send information to Bob's trusted platform, even though the attacker intercepts them on a different machine.

The IDSCP protocol is vulnerable because it includes public keys in the quote. Attestation protocols using only randomly drawn nonces as qualifying data, such as [2,6,9], are not vulnerable to this attack. However, these protocols have drawbacks that make them unsuitable for use cases such as the Industrial Data Space (c.f. Sect. 2.2). Hence there is still a need for a simple and easily usable remote attestation protocol that is not susceptible to this type of attack.

3 A Secure Protocol

A secure protocol needs to conduct a remote attestation and establish encrypted channels between the endpoints. On their own, both of these tasks have been solved. The problem we face when designing an attestation protocol is how to bind the keys responsible for the encrypted channel to the attested trusted platform. Doing this right is vital for a secure attestation protocol, because it ensures that only the legitimate trusted platform can read data transmitted over the attested channel. As we showed in the previous section, IDSCP is vulnerable to active man-in-the-middle attacks because it includes security relevant information such as the public part of a Diffie-Hellman key in the quote. A possible solution to this problem is to extend the PCRs with that information prior to attestation, instead of using it directly as qualifying data for the quote. Doing so shows a verifier that the Diffie-Hellman key pair used for key establishment was in fact generated on the trusted platform itself, while avoiding the vulnerability against relay attacks. However, this solution is not feasible in practice. Since each established channel requires a new key exchange, the legitimate PCR values will change constantly. As a result the measurement logs that verifiers

have to check become large and extremely cluttered. This makes it very difficult to update and maintain a trusted software stack in a dynamic and distributed environment. Another approach for binding the ephemeral public keys to the trusted platform is to sign them with the attestation identity key (AIK). Since only the TPM can sign with the AIK, this proves that the key originates from the trusted platform. Also it prevents relay attacks, since the key exchange is completely independent of the quote. However, the AIK is a restricted key and cannot be used for signing external data. It can only sign data structures that have been created by the TPM, such as quotes. This property of the AIK is in fact very important for the remote attestation, since otherwise the platform could use the TPM to certify a fake quote (e.g. with wrong PCRs). While this makes an AIK-certified key exchange impossible with the now outdated TPM version 1.2, there are more options available with a recent TPM in version 2.0. By implementing a TPM-managed key exchange, we can sign ephemeral keys with the AIK, thereby binding the key establishment to the trusted platform.

3.1 Protocol Design

A secure remote attestation protocol has to protect against internal attackers with access to platform secrets. However, TLS by design does not consider this type of adversary. As previously shown, combining the TLS-based encryption with attestation conflates security goals and complicates protocol applications. To avoid this we choose not to use TLS as underlying protocol. Instead we concentrate on establishing a shared secret that is guaranteed to be available only to the attested platform. This secret can then be used for manual encryption or as a key for an independent TLS layer. We assume that prior to the start of the protocol both participants have taken ownership of their TPM and created a storage root key. Furthermore they should have performed the AIK enrollment process and have their AIK certificates signed by a privacy CA. Our attestation protocol consists of three phases, which are shown in Table 1. The semantic of the functions as well as the parameter names follow the TPM 2.0 specification.

Initiation. During the *initiation phase*, both endpoints first create a TPM key template for appropriate encryption keys, which will be used in a later phase. Then random nonces are created and exchanged (steps 1 and 2). While the size of the nonces is not specified, it is recommended to use at least 160 bits of random data to ensure proper quote freshness. Furthermore, both sides transmit a selection of PCR numbers that they expect to be included in the quote.

Attestation. Afterwards the *attestation phase* is responsible for exchanging and verifying quotes. This is done in accordance with the TCG trusted attestation protocol [15]. For the sake of simplicity Table 1 only shows the explicit attestation. Nevertheless the other specified attestation types may be used as well. Since this affects only the TPM function calls and the contents of the quote, but

Table 1. Our proposed attestation protocol

	Initiation phase	
A, B	: $dhTemplate \leftarrow$ TPMT_PUBLIC(decrypt, KEY_SCHEME_ECDH)	
A	: Create a non-predictable nonce N^A	
$A \rightarrow B$:	$N^A, PCRSel^A$	(1)
B :	Create a non-predictable nonce N^B	
$A \leftarrow B$:	$N^B, PCRSel^B$	(2)
	Attestation phase	
A	: $(quoted^A, quoteSig^A) \leftarrow$ TPM2_Quote($akHandle^A, N^B, PCRSel^B$)	
$A \rightarrow B$:	$PCR^A, (quoted^A, quoteSig^A), akCert^A$	(3)
B :	Verify CA signature of $akCert^A$	
B :	Verify $quoteSig^A$ is a valid signature of $quoted^A$ under $akCert^A$	
B :	Verify $quoted^A$ contains expected PCR^A and N^B	
B :	$(quoted^B, quoteSig^B) \leftarrow$ TPM2_Quote($akHandle^B, N^A, PCRSel^A$)	
$A \leftarrow B$:	$PCR^B, (quoted^B, quoteSig^B), akCert^B$	(4)
A	: Verify CA signature of $akCert^B$	
A	: Verify $quoteSig^B$ is a valid signature of $quoted^B$ under $akCert^B$	
A	: Verify $quoted^B$ contains expected PCR^B and N^A	
	Key establishment phase	
A	: $dh^A \leftarrow$ TPM2_Create($srkHandle^A, dhTemplate$)	
A	: $dhHandle^A \leftarrow$ TPM2_Load($srkHandle^A, dh^A.private, dh^A.public$)	
A	: $(dhCertInfo^A, dhSig^A) \leftarrow$ TPM2_Certify($dhHandle^A, akHandle^A, N^B$)	
$A \rightarrow B$:	$dh^A.public, (dhCertInfo^A, dhSig^A)$	(5)
B :	Verify $dhSig^A$ is a valid signature of $dhCertInfo^A$ under $akCert^A$	
B :	Verify $dhCertInfo^A$ contains expected $dh^A.public$ and N^B	
B :	$dh^B \leftarrow$ TPM2_Create($srkHandle^B, dhTemplate$)	
B :	$dhHandle^B \leftarrow$ TPM2_Load($srkHandle^B, dh^B.private, dh^B.public$)	
B :	$(dhCertInfo^B, dhSig^B) \leftarrow$ TPM2_Certify($dhHandle^B, akHandle^B, N^A$)	
B :	$Z \leftarrow$ TPM2_ECDH_ZGen($dhHandle^B, dh^A.public$)	
$A \leftarrow B$:	$dh^B.public, (dhCertInfo^B, dhSig^B)$	(6)
A	: Verify $dhSig^B$ is a valid signature of $dhCertInfo^B$ under $akCert^B$	
A	: Verify $dhCertInfo^B$ contains expected $dh^B.public$ and N^A	
A	: $Z \leftarrow$ TPM2_ECDH_ZGen($dhHandle^A, dh^B.public$)	
$A \leftrightarrow B$:	Encrypt messages using secret $k := KDF(Z)$	

not the transmitted messages, our protocol is agnostic about the concrete attestation type. With explicit attestation, both participants execute TPM2_Quote in order to create a quote that is signed by the TPM with the attestation identity key. Only the previously received nonce is included as qualifying data. Afterwards the quote, its signature, the values of the requested PCRs and the AIK certificate are transmitted to the verifier (steps 3 and 4). Once the attestation information has been passed to the remote side, the AIK certificate, quote signature and quote information are being verified in that order. Since the protocol supports mutual attestation, both parties have to perform these protocol steps.

If any of the verification steps fail, either of the endpoints terminates the protocol handshake. After the attestation phase has been completed successfully, both endpoints have verified the integrity of the other trusted platform.

Key Establishment. Finally the *key establishment phase* establishes a shared secret between the two endpoints that is bound to the attested trusted platforms. For this we conduct a TPM-managed Elliptic Curve Diffie-Hellman (ECDH) key exchange. The TPM 2.0 specification conveniently offers functions for one-pass and two-pass key exchange protocols. Since we only want to conduct an ephemeral key exchange, the one-pass version is sufficient. Usually the one-pass key exchange is conducted asymmetrically using TPM2_ECDH_KeyGen. However, this is not feasible for our protocol, because we need to sign both ephemeral public keys with the restricted AIK. Because the TPM2_ECDH_KeyGen function directly outputs the generated ephemeral public point without proof of TPM ownership, we cannot use the AIK to sign this key. Also, in order to keep the protocol implementation as simple as possible, it is beneficial to perform a symmetric key establishment where both sides execute the same steps.

To solve these problems we conduct the key exchange by creating both ephemeral keys with TPM2_Create and invoking the TPM2_ECDH_ZGen function twice (c.f. Table 1). At first Alice creates a new ephemeral ECDH key pair by invoking the TPM2_Create method with the ECDH key template created earlier. It is important that this key is generated as a wrapped sub key instead of a root key using TPM2_Create_Primary. This is because primary keys are derived from the seldom changing platform secrets, while the ephemeral keys need to be randomly drawn for each handshake. After the ephemeral key pair has been created, it needs to be loaded into the TPM using TPM2_Load. Since the key pair has been generated in the TPM instead of the CPU, its public part can then be signed by the AIK using the TPM2_Certify function It is important to include the previously exchanged nonces into this signature, because otherwise the key exchange may be vulnerable to replay attacks. Finally Alice sends the public part of the ephemeral key, as well as the certification information and the signature to Bob (step 5). Bob can check if the received ephemeral key was in fact generated by the trusted system by verifying the signature with Alice's AIK certificate. He also checks if his nonce is included in the certification information. Then Bob uses TPM2_Create and TPM2_Certify to generate his own ephemeral ECDH key and AIK signature in the same way. By invoking TPM2_ECDH_ZGen with a handle on his own ECDH key and Alice's public key, Bob uses his TPM to calculate the shared secret Z. Finally he transmits his ephemeral public key and the corresponding signature back to Alice (step 6). After receiving Bob's public key and signature, Alice performs the same steps to verify the key and calculate the shared secret on her end. If any of the verification steps fail, either of the endpoints terminates the protocol handshake. Otherwise the protocol handshake completes with the establishment of the shared secret. Usually a symmetric encryption key is derived from the shared secret via a key derivation function (KDF), which protects the established channel. The nature

of the KDF and the subsequent encryption is not specified in the protocol handshake. We furthermore do not have any specific requirements for the network protocol that is used to send and receive the messages shown in Table 1.

3.2 Protocol Implementation

A reference implementation of our protocol is available (see footnote 2). To access TPM 2.0 devices we use Microsoft's TPM Software Stack. Since this software stack interfaces both physical TPM devices and a TPM 2.0 simulator, it is suitable for development purposes as well as productive use. As there are bindings for other programming languages, the protocol can be adopted easily for other platforms. We create the ephemeral ECDH keys using the NIST P-256 elliptic curve and SHA-256 message digests. The attestation handshake and all subsequent communication is conducted over the standard WebSocket protocol. After a successful handshake the symmetric channel encryption is achieved using the Java Crypto API and an AES-256 encryption with PKCS#7 padding.

4 Evaluation

In this section we analyze the presented protocol in terms of the previously defined security and functional requirements. We also compare our approach to previously proposed attestation protocols. Finally show how we verify our protocol using the Tamarin protocol verifier.

4.1 Security Analysis

In Sect. 2.1 we defined seven security goals S1 to S7. Now we show that our proposal fulfills all of these requirements. The security goals of authentication (S1) and platform integrity verification (S2) are satisfied by the attestation phase of the protocol. This phase performs a standard remote attestation of both endpoints by exchanging quotes that are signed with the platforms' attestation identity keys (AIK). Even an insider attacker with access to long-term secrets cannot forge an AIK signature, because the private key is only accessible to the TPM. Hence the AIK signature ensures that the verifier actually communicates with the correct platform, which fulfills the authentication requirement (S1). Naturally the exchanged quotes also attest to the integrity of the trusted platforms (S2), because they contain the requested selection of PCR values for the verifiers to check. Furthermore our protocol offers a secure key establishment (S3) by performing a Diffie-Hellman key exchange in the final phase of the handshake. Unlike with the proposal of Stumpf [14] and its variants, our key establishment is authenticated by the AIK. This enables the verifier to check that the ephemeral keys have in fact been generated on the authenticated trusted platform in the already attested state. As a result, our protocol is not vulnerable against the attacks defined by S4 to S7. The nonces that are exchanged during the initiation phase protect against replay attacks (S4). The protocol is also not vulnerable

against insider attacks (S5), because nothing is signed or encrypted with a non-TPM key that an insider attacker could use to his advantage. Especially the ephemeral keys are not signed with external long-term secrets, but are instead authenticated by the AIK, which is not available to the insider attacker. The main advantage of our protocol over IDSCP is its resilience against relay attacks even with a second attestation endpoint (S6 and S7). The key difference is that our protocol uses only nonces as qualifying data for the quote, not the hash of other security critical information. This prevents the presented relay attack with an additional attestation endpoint. A further advantage of our protocol is that the private ephemeral keys are generated by the TPM and never leave it. Therefore our protocol is not vulnerable against side-channel attacks on the CPU. Protocols that generate the ephemeral keys on the processor need to trust that they are not being disclosed, which is a severe disadvantage over creating them in the TPM. Table 2 compares the security properties of our protocol with previous proposals discussed in Sect. 2.2. For better comparison of vulnerable protocols, the second column shows the weakest attacker that successfully breaks one of the security requirements. We distinguish between an insider attacker (I), who knows non-TPM secrets, and an external adversary (E). Also the adversary can either be passive (P) or an active Dolev-Yao attacker (A). This notation gives an active insider attacker (IA) as strongest adversary, while a passive external attacker (EP) is the weakest adversary. The checkmark after the attacker properties shows if the attack is transparent (✓) or detectable by one of the protocol participants (✗). In summary, our protocol as well as the proposals of Armknecht and Lan are secure against all presented attacks. However, as already discussed in Sect. 2.2, in practice the latter protocols have disadvantages both in terms of security and functionality.

Table 2. Comparison of security properties.

Proposal	Attacker	S1	S2	S3	S4	S5	S6	S7	Remarks
Sailer [13]	EA (✗)	✗	✓	–	✓	–	–	–	No key exchange
Coker [5]	–	✓	✓	–	✓	–	–	–	No key exchange
Goldman [7]	IP (✓)	✓	✓	✗	✓	✗	✗	✗	PFS not mentioned
Cheng [4]	IP (✓)	✓	✓	✗	✓	✗	✗	✗	
IDSCP	IP (✓)	✓	✓	✗	✓	✗	✗	✗	
Aziz [3]	IA (✗)	✓	✓	✗	✓	✗	✗	✗	PFS not mentioned
Stumpf [14]	EA (✓)	✓	✓	✗	✓	✓	✗	✗	
IDSCP (PFS)	IA (✗)	✓	✓	✗	✓	✓	✓	✗	
Zhou [16]	IA (✗)	✓	✓	✗	✓	✗	✓	✗	
Armknecht [2]	–	✓	✓	✓	✓	✓	✓	✓	Only TPM 1.2
Lan [9]	–	✓	✓	✓	✓	✓	✓	✓	
Our protocol	–	✓	✓	✓	✓	✓	✓	✓	**Only TPM 2.0**

4.2 Protocol Properties

Considering the functional requirements, our protocol clearly supports mutual attestation (F1). Since the protocol explicitly separates attestation from key establishment, re-attestation is possible without a key change (F2). Finally, conducting a new ephemeral key exchange for each protocol handshake gives the forward secrecy property (F3). In terms of overhead our approach is of a similar complexity as IDSCP (F4). Our reference implementation has about 1700 lines of code, while IDSCP has about 2000. Furthermore, we have no link with a TLS protocol instance and are independent of the network stack. On the other hand, since our protocol calculates the key exchange directly on the TPM hardware instead of the much faster CPU, a performance impact is to be expected. Table 3 compares the functional properties of our protocol to the previous proposals.

Table 3. Comparison of functional properties.

Proposal	F1	F2	F3	F4	Remarks
Sailer [13]	✗	–	✗	Low	
Coker [5]	✗	–	✗	Medium	
Goldman [7]	✗	✓	(✓)	High	
Cheng [4]	✗	–	✗	Low	
IDSCP	✓	✗	✗	Low	
Aziz [3]	✓	✗	(✓)	Medium	
Stumpf [14]	✗	✗	✓	Low	
IDSCP (PFS)	✓	✗	✓	Low	
Zhou [16]	✓	✗	✓	Medium	
Armknecht [2]	✓	✓	✓	High	Only TPM 1.2
Lan [9]	✗	✗	✓	High	
Our protocol	✓	✓	✓	**Medium**	**Only TPM 2.0**

4.3 Formal Verification

In addition to the informal security analysis, we also verified our protocol using the Tamarin prover [11]. Our main goal in doing so is to show that – unlike IDSCP – our protocol is secure even with an additional attestation endpoint (security goal S7). First we modeled both our protocol and IDSCP with Tamarin's Diffie-Hellman equational theory. Then we defined the security requirements as Tamarin trace properties quantifying over the attacker's knowledge. These trace properties determine that for a secure protocol the attacker must not learn the established shared secret. We also added an additional rule that allows the attacker to retrieve signed quotes with any qualifying data.

This models the additional attestation endpoint on the trusted platform. If the rule is active, Tamarin in fact finds the man-in-the-middle attack on IDSCP we presented in Sect. 2.3. In contrast, Tamarin correctly verifies all security properties on our protocol, even with the additional attestation endpoint available to the attacker. All theorem definitions of both IDSCP and our protocol are available for verification (see footnote 2).

5 Conclusion

In this work we evaluated existing attestation protocols in terms of security and functionality. We found that many existing protocols are either vulnerable against insider or relay attacks, or depend on outdated TPM specifications. Furthermore we showed that an actively used remote attestation protocol (IDSCP) is vulnerable against a variant of relay attacks that assumes the existence of additional attestation endpoints. In response to this threat we proposed and analyzed a lightweight remote attestation protocol that is not vulnerable against this type of attack. Our protocol is based on the TPM 2.0 specification and offers mutual attestation, re-attestation and shared key establishment with perfect forward secrecy. Unlike previous proposals, our protocol generates secret keys exclusively inside the TPM, which protects against side-channel attacks on the CPU. Keeping our protocol completely independent of the underlying network stack makes it very flexible and easy to adapt for any use case specific requirements. Finally we analyzed the security of our protocol both informally as well as by modeling its security properties with the Tamarin theorem prover. A ready-to-use protocol implementation is publicly available as well (see footnote 2).

As future work we plan to implement our protocol for other platforms and more TPM software stacks. We also intend to evaluate the performance of our protocol more rigorously in the Industrial Data Space as an actual productive use case. Furthermore our protocol can be extended to also support direct anonymous attestation (DAA).

A The Industrial Data Space Communication Protocol

Table 4 illustrates the messages sent during an IDSCP handshake. A complete description of the IDSCP protocol is given in [10]. The reference implementation of IDSCP is available on Github (see footnote 1).

Table 4. IDSCP remote attestation protocol

	TLS handshake	
$A \leftrightarrow B:$	Establish a TLS channel with certificates $cert^A$ and $cert^B$	
	Initiation phase	
$A \rightarrow B:$	Non-predictable nonce N^A	(1)
$A \leftarrow B:$	Non-predictable nonce N^B	(2)
	Attestation phase	
A	: $(quoted^A, quoteSig^A) \leftarrow \text{TPM\_Quote}(akHandle^A, SHA1(N^B, cert^B))$	
$A \rightarrow B:$	$PCR^A, (quoted^A, quoteSig^A), akCert^A$	(3)
B	: $(quoted^B, quoteSig^B) \leftarrow \text{TPM\_Quote}(akHandle^B, SHA1(N^A, cert^A))$	
$A \leftarrow B:$	$PCR^B, (quoted^B, quoteSig^B), akCert^B$	(4)
	Verification phase	
A	: Verify CA signature of $akCert^B$	
A	: Verify $quoteSig^B$ is a valid signature of $quoted^B$ under $akCert^B$	
A	: Verify $quoted^B$ contains expected PCR^B and $SHA1(N^A, cert^A)$	
B	: Verify CA signature of $akCert^A$	
B	: Verify $quoteSig^A$ is a valid signature of $quoted^A$ under $akCert^A$	
B	: Verify $quoted^A$ contains expected PCR^A and $SHA1(N^B, cert^B)$	

References

1. Akram, R.N., Markantonakis, K., Mayes, K., Bonnefoi, P.F., Sauveron, D., Chaumette, S.: An efficient, secure and trusted channel protocol for avionics wireless networks. In: 35th Digital Avionics Systems Conference, pp. 1–10. IEEE (2016)
2. Armknecht, F., et al.: An efficient implementation of trusted channels based on OpenSSL. In: 3rd ACM Workshop on Scalable Trusted Computing, pp. 41–50 (2008)
3. Aziz, N., Udzir, N., Mahmod, R.: Extending TLS with mutual attestation for platform integrity assurance. J. Commun. **9**(1), 63–72 (2014)
4. Cheng, S., Bing, L., Yang, X., Yixian, Y., Li, Z., Han, Y.: A security-enhanced remote platform integrity attestation scheme. In: 5th Conference on Wireless Communications, Networking and Mobile Computing, pp. 1–4. IEEE (2009)
5. Coker, G., et al.: Principles of remote attestation. Int. J. Inf. Secur. **10**(2), 63–81 (2011). https://doi.org/10.1007/s10207-011-0124-7
6. Gasmi, Y., Sadeghi, A.R., Stewin, P., Unger, M., Asokan, N.: Beyond secure channels. In: 2007 ACM workshop on Scalable trusted computing, pp. 30–40 (2007)
7. Goldman, K., Perez, R., Sailer, R.: Linking remote attestation to secure tunnel endpoints. In: 1st ACM workshop on Scalable trusted computing, pp. 21–24 (2006)
8. Greveler, U., Justus, B., Löhr, D.: Mutual remote attestation: enabling system cloning for TPM based platforms. In: Meadows, C., Fernandez-Gago, C. (eds.) STM 2011. LNCS, vol. 7170, pp. 193–206. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29963-6_14
9. Lan, A., Han, Z., Zhang, D., Jiang, Y., Liu, T., Li, M.: An anonymous remote attestation protocol to prevent masquerading attack. In: 11th International Conference on Autonomic and Trusted Computing, pp. 590–595. IEEE (2014)

10. Lux, M., Brost, G.: The industrial dataspace communication protocol. A protocol for remote attestation and secure data exchange (2018)
11. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN prover for the symbolic analysis of security protocols. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 696–701. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_48
12. Otto, B., Lohmann, S., Steinbuß, S., Teuscher, A.: IDS reference architecture model. Technical report, International Data Spaces Association (2018)
13. Sailer, R., Zhang, X., Jaeger, T., Van Doorn, L.: Design and implementation of a TCG-based integrity measurement architecture. In: USENIX Security symposium, vol. 13, pp. 223–238 (2004)
14. Stumpf, F., Tafreschi, O., Röder, P., Eckert, C., et al.: A robust integrity reporting protocol for remote attestation. In: Proceedings of the Workshop on Advances in Trusted Computing (WATC), p. 65. Citeseer (2006)
15. TCG: Trusted attestation protocol (TAP) information model. Technical report (2019)
16. Zhou, L., Zhang, Z.: Trusted channels with password-based authentication and TPM-based attestation. In: 2010 International Conference on Communications and Mobile Computing, vol. 1, pp. 223–227. IEEE (2010)