Introducing Remote Attestation and Hardware-based Cryptography to OPC UA

Pascal Birnstill*, Christian Haas*, Daniel Hassler† and Jürgen Beyerer* †Fraunhofer Institute of Optronics, System Technologies and Image Exploitation IOSB, Karlsruhe, Germany
Email: {pascal.birnstill | christian.haas | juergen.beyerer}@iosb.fraunhofer.de †CAS Software AG, Karlsruhe, Germany
Email: daniel.hassler@cas.de

Abstract—In this paper we investigate whether and how hardware-based roots of trust, namely Trusted Platform Modules (TPMs) can improve the security of the communication protocol OPC UA (Open Platform Communications Unified Architecture) under reasonable assumptions, i.e. the Dolev-Yao attacker model. Our analysis shows that TPMs may serve for generating (RNG) and securely storing cryptographic keys, as cryptocoprocessors for weak systems, as well as for remote attestation. We propose to include these TPM functions into OPC UA via so-called ConformanceUnits, which can serve as building blocks of profiles that are used by clients and servers for negotiating the parameters of a session. Eventually, we present first results regarding the performance of a client-server communication including an additional OPC UA server providing remote attestation of other OPC UA servers.

I. INTRODUCTION

For several decades production systems were considered as closed and decoupled units, where information and network security has not been an issue. This is changing rapidly in the course of the so-called fourth industrial revolution (Industry 4.0), which is characterized by globally distributed and highly flexible value and supply chains. These are orchestrated beyond enterprise networks, using the Internet so as to enable new higher-level services and business models. This evolution towards smart factories is accompanied by the application of the paradigms of the Internet of Things and Services (IoTS) to production systems and assets. The result is a mash-up of office and production IT networks inheriting the security challenges of office IT networks connected over the Internet.

These challenges were been taken into account when the Open Platform Communications Unified Architecture (OPC UA) protocol was designed and standardized by the OPC foundation. OPC UA is a multi-layer client-server communication protocol for industrial automation systems based on a service-oriented architecture (SOA). OPC UA is already capable of establishing secure communication channels and somehow seems to have been inspired by the widespread Transport Layer Security protocol (TLS). However, the utilization of hardware-based trust anchors, such as Trusted Platform Modules (TPMs) has not yet been considered although these modules offer capabilities to improve existing security features, such as key generation and storage, and also to cover further aspects of security, i.e., remote attestation of a system’s integrity state. Via remote attestation, communication partners can reassure their peers that their systems are still in a state which was considered trustworthy at some point in time.

Our contribution is an integration of TPM-based cryptographic functions into the OPC UA specification with particular focus on remote attestation. For this security mechanism we provide a first performance evaluation as well as a security analysis, which shows that remote attestation can improve the security of OPC UA client-server communication under the assumptions of the Dolev-Yao attacker model [1].

This paper is structured as follows. In section II we introduce OPC UA with focus on its security-related features. Section III explains trusted computing mechanisms based on TPMs that can further improve the security level provided by OPC UA, namely remote attestation, random number generation, key generation and storage, as well as using a TPM as a cryptocoprocessor for computationally weak devices. Subsequently we discuss how these mechanisms could be integrated into the OPC UA specification based on so-called ConformanceUnits. We present an implementation of remote attestation, for which we provide a performance evaluation as well as a security analysis in section V. Eventually, we discuss related work in section VI and conclude in section VII.

II. OPC UA

Open Platform Communications Unified Architecture (OPC UA) is a client-server communication protocol of the OPC Foundation specified in the standard 62541 of the International Electrotechnical Commission (IEC). It provides a standardized software interface for exchanging data among applications of different manufacturers of automation technology, which follows the SOA principle (service-oriented architecture). As a revision of the Open Platform Communications (OPC) protocol, it includes improvements regarding platform independence, better support for security mechanisms, and an object-oriented information model on which services based on OPC UA can be mapped [2]. For the sake of brevity, our description of OPC UA focuses on security aspects.

A. Object-Oriented Information Model of OPC UA

Each OPC UA servers provides specific information and functions to OPC UA clients. These capabilities are modeled
using OPC UA’s object-oriented information model. An OPC UA server’s specific set of information and functions is denoted as its address space. Prior to being able to search an OPC UA server’s address space and to employ its services a client needs to establish a communication relationship with this server. For this, the client requests the so-called EndpointDescriptions of the server. This is where OPC UA’s security mechanisms set in.

B. Security Mechanisms of OPC UA

An EndpointDescription contains information required for establishing a SecureChannel, i.e., the logical connection between a client and a server communicating over OPC UA. In case a server returns several EndpointDescriptions, it is up to the client to decide on one of them.

OPC UA defines three operational modes for SecureChannels. NONE equals no security, i.e., transmitted messages are neither signed nor encrypted, SIGN means the messages are signed, and SIGNANDENCRYPT requires both, signing and encrypting of messages [3]. The cryptographic algorithms for a SecureChannel are negotiated via SecurityPolicy Profiles, in which clients and servers specify the security mechanisms they support. For this, the attribute securityPolicyUri of an EndpointDescription specifies the SecurityPolicy Profile to be used. Since security mechanisms have to be updated from time to time, e.g., if vulnerabilities are discovered or to support larger key lengths, SecurityPolicyProfiles are extensible. We show the SecurityProfiles of the current standard in table I. For the keys for asymmetric algorithms, X.509 certificates in version 3 have to be used.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Signature</th>
<th>Asym. Encrytion</th>
<th>Sym. Enc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Basic128Rsa15</td>
<td>SHA-1</td>
<td>RSA (1024-2048 Bit)</td>
<td>AES (128 Bit)</td>
</tr>
<tr>
<td>Basic256</td>
<td>SHA-1, SHA-256</td>
<td>RSA (1024-2048 Bit)</td>
<td>AES (256 Bit)</td>
</tr>
<tr>
<td>Basic256Sha256</td>
<td>SHA-256</td>
<td>RSA (2048-4096 Bit)</td>
<td>AES (256 Bit)</td>
</tr>
</tbody>
</table>

SecureChannels are established using the OPC UA Secure Conversation protocol (UASC), which is inspired by the widespread Transport Layer Security protocol (TLS). During the handshake for establishing a SecureChannel, as in TLS, the client and the server exchange nonces, i.e., random numbers which protect against replay attacks and which are also used for deriving session keys for encrypting the communication. The quality of these random numbers is thus crucial for the attained level of security.

The concept of Profiles is introduced in part 7 of the OPC UA specification [4]. They describe the capabilities of an OPC UA client or server, e.g., support basic services or security mechanisms. Profiles belong to one of the categories Client, Server, Transport, and Security. They consist of so-called ConformanceUnits or further Profiles, where a ConformanceUnit describes a certain capability and according test cases. ConformanceUnits are associated to ConformanceGroups, e.g., Security. Table II shows the SecurityPolicy Profile Basic256.

### Table II

**SecurityPolicy Profile Basic256 [4]**

<table>
<thead>
<tr>
<th>Group</th>
<th>ConformanceUnit / Profile</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security</td>
<td>Basic 256</td>
<td>X</td>
</tr>
<tr>
<td>Security</td>
<td>Certificate Validation</td>
<td>X</td>
</tr>
<tr>
<td>Security</td>
<td>Encryption Required</td>
<td>X</td>
</tr>
<tr>
<td>Security</td>
<td>Signing Required</td>
<td>X</td>
</tr>
</tbody>
</table>

III. TRUSTED PLATFORM MODULES (TPM)

The Trusted Computing Group (TCG) issues various specifications and standards for hardware-based roots of trust so as to enable interoperable trusted computing platforms. Key standards of the TCG therefore describe Trusted Platform modules (TPM), which are dedicated hardware units providing security-related functions. TPMs are based on microcontrollers that autonomously compute cryptographic functions without relying on any component of the enclosing platform and hence are hard to attack.

We examined the specifications for TPM 1.2 [5] and TPM 2.0 [6] for functions to improve the security of OPC UA. The following functions have been identified and will be explained in the subsequent paragraphs:

- Remote attestation
- Random number generation (RNG)
- Key generation and storage
- Usage as cryptocompressor

A. Remote Attestation

Remote attestation is about establishing trust in a system’s integrity. Before sharing data with or fetching data from a remote system, one may want to make sure that this system is still in a condition, which was considered trustworthy at some point in time, e.g., because it has been formally verified or certified by a trusted authority. Remote attestation therefore requires that the integrity of all relevant components is checked before they are used or executed. This process already sets in once the considered system is switched on. If and only if we can trust in each particular step of a system’s start-up procedure, we can ensure that the system as a whole can be trusted in. So each step in this so-called chain of trust requires trust in the previous step. As the first element cannot be validated by a predecessor, it must be a component that is inherently trustworthy [7]. This role of a trust anchor can be occupied by a TPM, which is trusted due to the open specification of the TCG as well as the undertaken certification procedures. The involved components within the TPM specification are the Root of Trust for Measurement (RTM), Root of Trust for Storage (RTS), and Root of Trust for Reporting (RTR). The RTS is the secure storage area of the TPM, which is protected against unauthorized access. The RTM stores information required for platform integrity checking in the RTS. When booting up the system, the first instructions to be executed by the TPM are called Core Root of Trust for Measurement (CRTM). These instruction take measurements for checking the integrity of the platform, which usually includes the firmware (e.g., the BIOS) and
its configuration. These measurements are written into the designated Platform Configuration Registers (PCR) of the RTS. Similarly, in the next steps the integrity of the boot loader, (relevant components of) the operating system and selected applications are checked and the PCRs are extended accordingly. Eventually the PCRs unambiguously reflect the given system's state in terms of integrity. This chain of trust is also depicted in figure 1.

The RTR is in charge of reporting about the system’s integrity. It reads the PCRs and adds a cryptographic signature to authenticate the measurements. Such signatures are computed using so-called Attestation Identity Keys (AIK). AIKs are RSA key pairs generated by the TPM, which are designated for creating such authenticated fingerprints of the according system. AIKs are stored outside the TPM in an encrypted fashion and can only be decrypted and used within the TPM. Thus an arbitrary number of AIKs can created. However, the authenticity of an AIK must be confirmed by a trusted third party.

B. Random Number Generation

Generating cryptographic keys usually requires high-quality random numbers. Serving as root of trust, a TPM cannot rely on external random number generators (RNG). RNGs of TPMs typically mix internal random data with additional sources of entropy such as clock instabilities, thermal radiation, or keyboard/mouse input [5].

C. Key Storage

TPMs are equipped with a small-size non-volatile memory for storing cryptographic keys. However, using a TPM’s internal Storage Root Key (SRK) an arbitrary number of keys can be stored securely in external memory. These so-called Storage Keys (SK) can again be used to generate further SKs, resulting in a hierarchy with the SRK as its root. All storage keys are asymmetric key pairs, where the private key is encrypted with the public key of a pair of the above hierarchy level. These encrypted private keys can be kept outside the TPM. For decryption they have to be loaded into the TPM.

D. Symmetric Encryption

In the TPM 1.2 standard encryption algorithms of a TPM are only used internally, i.e., there is no interface for using these functions from the outside [5]. This changed in the TPM 2.0 standard, according to which TPMS have to provide symmetric encryption for external usage.

IV. TRUSTED COMPUTING EXTENSIONS FOR OPC UA

Having identified the TPM functions that would be beneficial for the security of OPC UA (cf. section III), we need to discuss how these functions could be integrated into OPC UA. This is particularly important for extensions which have to be supported by both partners of a communication. From the considered TPM functions this applies to remote attestation, which also requires the presence of an entity serving as an attester for validating the system state as reported by the server (attester). The other TPM functions considered here do not require interaction.

Looking at EndpointDescriptions, the first option is to create an additional securityMode. However, as securityModes cannot be nested, we would have to define additional securityModes for each particular combination of security mechanisms. Another option is adding a new SecurityPolicyProfile. This approach is more flexible, since OPC UA allows nesting of Profiles (cf. section II-B). Eventually, a third option for signaling the remote attestation capability is using the additionalHeader attribute of the RequestHeader when establishing a SecureChannel. While it seems to be rather flexible to use, according to the specification it is reserved for future usage and is ignored by peers who do not understand or support it. This raises concerns regarding the compatibility with future versions of the OPC UA standard. We therefore argue in favor of using Profiles for specifying the TPM functions to be used in OPC UA. In the following we describe how the TPM extensions can be integrated into OPC UA and introduce the particular ConformanceUnits that we need to add.

A. Remote Attestation in OPC UA

As explained in section III-A an integration of remote attestation in OPC UA offers added value in terms of security as it adds the capability to check the integrity of a host system given a system integrity state that is considered secure. In the following we discuss how a remote attestation extension for OPC UA can be implemented.

Building a chain of trust upon a TPM we obtain an unforgeable checksum of a given system’s integrity state. However, since we cannot assume that all peers in our network know the trustworthy integrity states of each other, it is not sufficient that an OPC UA server publishes its integrity state. Instead we need to add the role of an attester, which holds the trustworthy integrity checksums of other systems. For this, a system’s trustworthy checksum must be submitted to the attester whenever a system is put into operation or updated. This needs to be done manually, i.e., by a trusted system administrator. Initially this also requires that an AIK of the system to be put into operation is generated using its TPM, and that this AIK is authenticated (e.g., signed) by a trusted
third party, i.e., a public key infrastructure (PKI), which can be made accessible to OPC UA nodes via the Global Discovery Service (GDS) acting in the role of a CertificateManager. The attester can be implemented as server, as client or as part of OPC UA’s GDS.

We also cannot assume that a server is capable of recognizing its own compromisation, since precisely the circumstance of being compromised may suppress this recognition (lying endpoint problem). Since an attacker may either be able to suppress the attacked system’s recognition of being compromised or the self-publication of this fact, it is not possible to make use of OPC UA’s concepts of Alarms and Conditions to publish critical changes of a server’s integrity state. An interaction between client and server is thus mandatory. This can be implemented as method invocation in the address space of the OPC UA server, the integrity of which is to be validated. In figure 2 this method is called AttestIntegrity. The client invokes this method with a nonce, which the server passes to the TPM function TPM Quote 2 to obtain a so-called quote. The quote includes a signature over the nonce and certain PCRs using an AIK (cf. section III-A) as well as the input values themselves. By means of the nonce we ensure that we obtain an up-to-date quote. After receiving the quote, the client needs to validate it against a trustworthy integrity state of the server, which can be requested from the attester via the method GetQuote. It returns a trustworthy quote of the server signed by the attester and should also be protected against replay attacks using another nonce. If one of the signatures is invalid or if the quotes do not match, the client must assume that the server has been compromised. In case the client is also equipped with a TPM, it can of course be used for validating the server’s and the attester’s signatures. As explained above, authenticated public keys of AIKs as well as of the attester can be obtained via OPC UA’s GDS.

The ConformanceUnit for an OPC UA server implementing remote attestation is shown in table III. Now the remaining question is about the occasions on which a client is supposed to request remote attestation. Methods of an OPC UA server can be invoked once a SecureChannel has been established, and the proposed ConformanceUnit (cf. table IV) accordingly demands that remote attestation is executed instantly afterwards. However, it is up to the client to make further request for remote attestation, e.g., before security-critical functions are used. For both ConformanceUnits we provide the test cases in the appendix (cf. section VII).

### Table III

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security/Server</td>
<td>Security/Client</td>
<td>The server provides the AttestIntegrity in its address space. The method shall be implemented as described in IV-A.</td>
</tr>
</tbody>
</table>

### Table IV

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security/Client</td>
<td>Security/Client</td>
<td>If the server implements the method described in III, then it must be invoked by the client instantly after the session with the server has been established. The client either needs to be able to perform the integrity validation by itself or with the help of an attester.</td>
</tr>
</tbody>
</table>

### Table V

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Practice - Security/Client</td>
<td>Random numbers used for security-relevant Hardware functions</td>
<td>Generated within a hardware based RNG, e.g., a TPM.</td>
</tr>
</tbody>
</table>

### B. Random Number Generation

Part 2 of the OPC UA specification argues against using the RNG rand() from the C standard library [8, para. 6.4]. OPC UA’s ConformanceUnit Best Practice – Random Numbers [4, para. 6.5.142] for random numbers requires that RNGs from crypto libraries are used so as to obtain random numbers with sufficient entropy. Nevertheless, a security analysis of German Bundesamt für Sicherheit in der Informationstechnik (BSI) criticizes that the OPC UA specification does not define concrete quality requirements for RNGs [9]. This criticism can be addressed by adding a new ConformanceUnit demanding the usage of a TPM for random number generation as shown in table V.

### C. Key Generation and Storage

As described in section II-B the OPC UA securityMode SIGNANDENCRIPT requires an asymmetric key pair. However, the OPC UA specification does not address the problem of protecting the private key, which is crucial for the security level of SecureChannels. We therefore suggest that if a TPM is available in a given system, then it should be used for keeping private keys secret. An according Best Practice ConformanceUnit is shown in table VI. In case a key pair is generated on the local system, this task could also be taken over by a TPM.
D. TPM as Cryptocoprocessor

For cryptographic functions OPC UA relies on the according software stack of the operating system. This is usually faster than using according functions of a TPM (TPM 2.0, cf. section III) and also no risk in terms security as long as we trust into the OS. On the other hand, there are also systems with low computation power in OPC UA networks. For such devices it would indeed be beneficial to employ a TPM for computationally expensive cryptographic functions. However, this is a purely implementation-related decision, which is not of interest for communication partners. Thus we do not need to add a specific Profile/ConformanceUnit for using a TPM as cryptocoprocessor.

OPC UA employs encryption if the SecureChannel is configured accordingly or if TLS is used as the transport layer security protocol. By default, both software stacks make use of the standard crypto libraries such as OpenSSL. Hence, adaptations of the according software stacks would be required in order to delegate cryptographic operations to a TPM. According operations provided by the TPM 2.0 specification are:

- Symmetric encryption and decryption (TPM2_EncryptDecrypt)
- Asymmetric encryption and decryption (TPM2_RSA_Encrypt, TPM2_RSA_Decrypt)
- Signing and verifying signatures (TPM2_SIGN, TPM2_VerifySignature)
- HMAC (TPM2_HMAC)

V. Evaluation

Our evaluation comprises a performance test of a prototypical implementation as well as security analysis. For the performance evaluation we implemented the AttestIntegrity method described in section IV-A for an OPC UA server, which offers this method in its address space. We also implemented the attester as an OPC UA server providing the GetQuote method in its address space, whereby we assume that the attester’s database already contains a trustworthy fingerprint for the server we want to validate. Eventually we implemented a client who is capable of validating quotes (Tp spi_TPM_VerifySignature). This validation includes checking the nonces, the fingerprints contained in the quotes, as well as the signatures of the server and the attester.

For obtaining a fingerprint of the system, we used the Integrity Measurement Architecture (IMA), which is part of the Linux kernel. It carries on building up the chain of trust while the system boots up, i.e., it computes a checksum of the runtime environment, which is then used to extend a PCR. The files to be contained in this measurement can be specified in IMA policies. By default, all files are included which are executable or readable by the root user. By this means any modification on one of the included files can be detected.

For accessing the TPM functions we used the TrouSerS\(^1\) library as well as the TPM tools for Linux, our OPC UA servers are based on the open source stack open62541\(^2\).

A. Performance Evaluation

For the performance evaluation we used two different systems with different TPMs (TPM specification 1.2) for the server to be attested, namely a ThinkPad T510 with an integrated ST Microelectronics TPM as well as a Raspberry Pi 2 with an infineon SLB9645 TPM. Server, client and attester were operated in a local network connected via a standard switch.

We measured the time consumption of the OPC UA methods AttestIntegrity and GetQuote, but also the pure processing time of the TPM functions. The time consumption of the OPC UA methods also includes the end-to-end delay of the network, since they are measured between the client’s request and the receipt of the server’s reply.

For measuring the performance of AttestIntegrity we performed 300 method invocations. We measured the time consumption of the OPC UA method at the client and the time consumption of the TPM function TPMQuote on the server (cf. figure 2). In the first iteration we run the server on the Raspberry Pi 2 and the client on the ThinkPad, for the second iteration we interchanged the roles. In a third iteration we bypassed the TPM so as to obtain measurements of the network delay for a comparison (note that the y-axis depicts the time in ms instead of s). Figure 3 shows the results as box-and-whisker plots. For iteration 1 50% of the measurements for Tspi_TPM_Quote were in between 1.1s and 1.6s. The results for AttestIntegrity were only marginally higher, which indicates that the computation of the quote in the TPM is the determining factor for the time consumption of this method. This observation is confirmed in iteration 2 and 3. Iteration 2 additionally shows that the older TPM hardware of the ThinkPad is considerably slower. In iteration 3 we observe that bypassing the TPM results in a time consumption of around 500ms with a few upward outliers.

We did not measure the time consumption of the GetQuote method, since the attester’s task is almost identical to the server’s task when executing AttestIntegrity, i.e., it needs to look up the trustworthy fingerprint for the server in its database and to calculate a signature over the fingerprint and the client’s nonce.

Eventually, we measured the time consumption for Tspi_Hash_Verify_Signature, i.e., the verification of the quotes’ signatures (cf. figure 2). A bit surprisingly, as we can see in figure 4 this operation is faster than Tspi_TPM_Quote2 by an order of magnitude and this time the ThinkPad is faster than the Raspberry Pi. This observation encourages the assumption that this operation is not executed within the

\(^1\)https://trousers.sourceforge.net/
\(^2\)https://open62541.org/
TPM. We can confirm this assumption as the TrouSerS source code calls RSA_verify from the OpenSSL library. This observation is also in line with the TPM 1.2 specification, however, the TPM 2.0 specification explicitly demands a function for verifying RSA signatures.

B. Security Analysis

For our security analysis we assume that the OPC UA server as well as the client are implemented and operated according to the ConformanceUnits described in section IV-A. We further assume that the server system is trustworthy with an established chain of trust, and that the private keys used for remote attestation are securely stored using the TPM. We consider an attacker with the following capabilities according to the Dolev-Yao model [1]:

- The attacker acts as an authorized participant of the network and is able to send messages to any other participant
- The attacker can spoof the identity of other participants in the network

Being able to discard any message of the AttestIntegrity method this attacker can of course fully suppress the execution of the remote attestation procedure described in this paper. In this case the client will assume that the server has been compromised and abandon the communication relationship. The same applies if the attacker discards the messages originating from the attester.

The attacker can neither spoof replies from the server nor from the attester, since these messages contain signatures including the quotes and random nonces. If, like in our setup, the attester is implemented as an OPC UA server, which at least demands the securityMode SIGN, then the SecureChannel will also ensure that its messages cannot be spoofed by the attacker. All in all, given that the client does not reuse nonces, the attacker cannot pretend trustworthy system states, i.e., he cannot perform replay attacks so as to fake quotes.

1) Attacks Against the OPC UA Server: Assuming that an attacker gains full control over the OPC UA server, he can infiltrate the system with malware. The goal of this attacker would be to conceal this compromise from the client so that he can, for instance, forge data that clients can request via some service. We assume that the attacker does not manage to deploy the malware without causing a change of the server’s PCRs, i.e., he still cannot break the chain of trust. As described above, suppressing the communication is not sufficient for deceiving the client. The remaining question is whether the attacker can succeed by either manipulating the OPC UA messages or the system.

PCR values can only be manipulated via the TPM_Extend operation. However, the PCR values cannot be set directly. They are extended using a cryptographic hash function, which the attacker would have to break in order to set arbitrary PCR values. As a consequence the attacker cannot prevent that the Tspl_TPM_Quote2 operation will return a quote with an unfavorable fingerprint revealing the system’s infiltration with malware. In case the attacker manipulates the quote, he needs to generate a valid signature. As this signature is calculated using an AIK whose private key can only be decrypted inside the TPM, the attacker would have to break the encryption algorithm in order to generate a valid signature for a faked quote.

2) Attacks Against the Attester and OPC UA Server: Once an attacker is able to gain control over the attester system and the OPC UA server at the same time, then remote attestation is meaningless, since the attacker can proceed as follows. The attacker deploys the malware on the OPC UA server. He then employs the atesters identity to request remote attestation from the server. The server’s reply can then be used to update the trustworthy fingerprint in the attester’s database. As soon as a client requests remote attestation from the OPC UA server, the attester will attest the compromised state of the server as trustworthy. Since the client relies on the atesters reply,
he must assume that both systems are in a trustworthy target state.

VI. RELATED WORK

The foundations of establishing a chain of trust to increase the trustworthiness of a system data back to the AEGIS concept introduced in 1996 [10]. The principle that we can only trust into a system if each particular step of its boot process can be trusted in was fed into the standards issued by the TCG according to which TPMs and trusted software stacks are built.

Stumpf et al. [11] proposed a two-step challenge-response procedure for remote attestation, which is robust against a masquerading attacker. A masquerading attacker assumes a false identity by means of a man-in-the-middle attack between the attestee and the challenger requesting remote attestation. This attack is effective if and only if remote attestation is performed via an insecure communication channel, e.g., if OPC UA employs the securityMode NONE. The attack can be prevented using securityModes SIGN and above.

In the IETF Internet draft [12] Fuchs et al. specify an unidirectional method for remote attestation, which is based on a trustworthy time source and a trusted time stamp authority. In the context of OPC UA we do not see the necessity of avoiding the challenge-response procedure for remote attestation, since communication is always between a client and a server and based on a previously established bidirectional session.

Jain and Vyas [13] integrated remote attestation into the TLS handshake. The authors also employed a TPM for the required signatures as well as for generating key material. Based on reasonable assumptions the paper shows that the proposed protocol is secure. This approach is comparable to ours, however, as we considered the OPC UA protocol, sessions tend to last longer. This is because sessions are often used for periodical method invocation or for calling several methods provided by a server. As a consequence, we also prescribe that remote attestation is performed during the initial handshake, but additionally allow for requesting remote attestation at arbitrary occasions during a session, e.g., before security-critical methods are executed.

VII. CONCLUSION

At the highest security level OPC UA uses authentication and encryption when transmitting messages. Using a TPM, these operations can be further hardened. The key material for signatures and encryption can be generated and stored within the TPMs secure storage. For weak devices the TPM can also serve as cryptographic functions can be outsourced to the TPM. Remote attestation, e.g., using TPMs as hardware-based trust anchors for estimating and authenticating integrity checksums of systems and/or particular software components considerably improves the security of OPC UA-based communication. By this means, we can validate at run-time whether a system’s current state still complies to a state, which has a-priori been considered trustworthy. Our analysis of OPC UA also showed that the proposed extensions could be specified in so-called security profiles, which are negotiated between communication partners. As expected, our performance evaluation showed that the TPM is indeed a bottleneck when establishing a session for OPC UA. Thus, once the software support improves, we expect that moving to TPMs in version 2.0 will improve the performance of the remote attestation procedure considerably.

In the medium run, an integration of hardware-based trust anchors into OPC UA may also serve as a door opener for more sophisticated security mechanisms such as modern access control or usage control mechanisms. In times of globally distributed supply chains and manufacturing chains, such mechanisms for increasing data sovereignty may work as additional catalysts for global collaboration.

ACKNOWLEDGMENT

This work was supported by the German Federal Ministry of Education and Research within the framework of the project KASTEL_SVI in the Competence Center for Applied Security Technology (KASTEL)."

REFERENCES


APPENDIX

A. Test Cases for Remote Attestation Method (Server)

Test 1: Invocation of AttestIntegrity with a nonce

Expected result: The server invokes the function TPM_Quote2 of its TPM. The TPM returns a quote and a signature including the nonce and the quote.

StatusCode = Good
Test 2: Invocation of AttestIntegrity without a nonce

Expected result: The server denies the request. The request is not processed. StatusCode = Bad_AttributeIdInvalid

Test 3: Invocation of AttestIntegrity with a nonce; server has no access to its TPM

Expected result: The invocation of TPM_Quote2 fails. The server is unable to answer the request. StatusCode = Bad_ResourceUnavailable

B. Test Cases for Remote Attestation Method Call (Client)

Test 1: After receiving the EndpointDescriptions from the server, the client selects an endpoint which supports the ConformanceUnit of table III (Security Remote Attestation Method). Subsequently the steps shown in figure 5 have to be performed.

Test 2: Invocation of AttestIntegrity with false signature

Expected result: AttestIntegrity does not fail, but result has not been signed correctly. The client aborts the session.

Test 3: Invocation of AttestIntegrity with false quote

Expected result: AttestIntegrity is executed successfully, but matching the quote with the trustworthy target value fails. The client aborts the session.

Test 4: Timeout of AttestIntegrity

Expected result: After the invocation of AttestIntegrity the client does not wait longer than 10s for a reply. In case of a timeout, the client aborts the session.