Towards Post-Quantum Security for Cyber-Physical Systems: Integrating PQC into Industrial M2M Communication[*]

SEBASTIAN PAUL[†], Robert Bosch GmbH, Germany
PATRIK SCHEIBLE, ESCRYPT GmbH, Germany
FRIEDRICH WIEMER, Robert Bosch GmbH, Germany

The threat of a cryptographically relevant quantum computer contributes to an increasing interest in the field of post-quantum cryptography (PQC). Compared to existing research efforts regarding the integration of PQC into the Transport Layer Security (TLS) protocol, industrial communication protocols have so far been neglected. Since industrial cyber-physical systems (CPS) are typically deployed for decades, protection against such long-term threats is needed.

In this work, we propose two novel solutions for the integration of post-quantum (PQ) primitives (digital signatures and key establishment) into the industrial protocol Open Platform Communications Unified Architecture (OPC UA): a hybrid solution combining conventional cryptography with PQC and a solution solely based on PQC. Both approaches provide mutual authentication between client and server and are realized with certificates fully compliant to the X.509 standard. We implement the two solutions and measure and evaluate their performance across three different security levels. All selected algorithms (Kyber, Dilithium, and Falcon) are candidates for standardization by the National Institute of Standards and Technology (NIST). We show that Falcon is a suitable option – especially – when using floating-point hardware provided by our ARM-based evaluation platform. Our proposed hybrid solution provides PQ security for early adopters but comes with additional performance and communication requirements. Our solution solely based on PQC shows superior performance across all evaluated security levels in terms of handshake duration compared to conventional OPC UA but comes at the cost of increased handshake sizes.

In addition to our performance evaluation, we provide a proof of security in the symbolic model for our two PQC-based variants of OPC UA. For this proof, we use the cryptographic protocol verifier ProVerif and formally verify confidentiality and authentication properties of our quantum-resistant variants.

Additional Key Words and Phrases: cyber-physical systems, post-quantum cryptography, formal security models, OPC UA, ProVerif.

1 INTRODUCTION

Google's recent shot at quantum supremacy attracted much public attention, but the road to a stable and large-scale quantum computer is still long and uncertain [7]. Once one is built, however, it will be able to solve mathematical problems previously thought to be intractable. As a consequence, public key primitives that have become the "security backbone" of our digital society will be broken. This threat can be mitigated by deploying new cryptographic primitives that withstand attacks from both quantum and traditional computers, i.e. post-quantum cryptography. NIST addressed this issue by starting a PQC standardization process in 2016, which is currently in its second round.

[*]This is an extended version of *"Towards Post-Quantum Security for Cyber-Physical Systems"* that originally appeared in *Computer Security – ESORICS 2020, Springer, pp. 295–316, DOI: 10.1007/978-3-030-59013-0_15.*
[†]Corresponding author.

Authors' addresses: Sebastian Paul, sebastian.paul2@de.bosch.com, Corporate Sector Research and Advance Engineering, Robert Bosch GmbH, Renningen, Germany; Patrik Scheible, patrik.scheible@escrypt.de, Consulting Cyber Security Solutions, ESCRYPT GmbH, Stuttgart, Germany; Friedrich Wiemer, friedrich.wiemer@de.bosch.com, Cross-Domain Computing Solutions, Robert Bosch GmbH, Stuttgart, Germany.

Eventually, NIST will standardize quantum-resistant key encapsulation mechanisms (KEMs) and digital signature algorithms (DSAs).[1]

A migration to new primitives requires various forms of cryptographic agility, which typically is not present in existing systems [52, 63]. Therefore, research how to securely and effectively integrate PQC into protocols and applications is required. Furthermore, it is essential to plan for the cryptographic transition, especially for devices with long life spans and high-security requirements. Several governmental institutes have proposed to use hybrid modes for this cryptographic transition [12, 28]. In such a hybrid mode at least two cryptographic primitives are applied simultaneously. On the one hand, a hybrid approach implies various advantages: 1) As long as one of the involved schemes remains unbroken, the "entire" security property holds. Therefore, early adopters can benefit from additional security against quantum adversaries but don't have to fully rely on relatively new primitives; 2) Being compliant to industrial or governmental standards that have not been updated yet to include PQC; 3) Provide backward compatibility to legacy devices. On the other hand, hybrid modes negatively affect performance and increase the required communication bandwidth as well as memory footprint.

One domain where components have long life spans and many industrial (or even governmental) regulations are in place are industrial control systems (ICS). In recent years, ICS have shifted away from isolated networks and serial communication towards highly connected networks and IP-based communication, ultimately, providing access to the Internet. In fact, modern industrial communication has shifted away from proprietary protocols towards standardized machine-to-machine (M2M) protocols such as OPC UA [57, 66, 77]. Taking into consideration that CPS deployed today could still be in use when a cryptographically relevant quantum computer is available, a migration plan towards PQC is highly recommended. Such a migration plan is even more critical regarding confidentiality because any communication passively recorded today can be retroactively decrypted once sufficiently powerful quantum computers become available. The fact that attacks related to industrial espionage play a major role in ICS further emphasizes the need for long-term confidentiality of transmitted data [75]. Although authentication can not be broken retroactively, we consider a preliminary investigation beneficial. As components of ICS are seldom updated during their long lifetime, they should support PQ DSAs rather sooner than later. As a consequence, we address the integration of PQC (KEM and DSA) into the widespread industrial communication protocol OPC UA in this work. Previous research efforts largely focused on the integration of PQC into common Internet protocols, mainly, concentrating on PQ key exchange. To the best of our knowledge, this is the first work that evaluates the integration of PQC into an industrial protocol.

1.1 Contribution

In this work, we integrate post-quantum means of key establishment and authentication into OPC UA's security handshake, demonstrating that industrial CPS are capable of handling the increased cost of PQC. Furthermore, we formally analyze the security of our proposed quantum-resistant variants in the symbolic model. The main contributions of our work are summarized as follows:

→ We investigate all lattice-based schemes of NIST's second round standardization process with regards to a security-size trade-off and conduct a standalone performance analysis of selected candidates on our evaluation platform.

[1]Our initial performance analysis conducted in [64] was based on algorithm specifications from the second round of NIST's standardization process. In February 2021, NIST released the third round candidates. Nevertheless, all selected algorithms (Kyber, Dilithium, and Falcon) are among the finalists. As submission teams were only allowed to slightly alter existing specifications, the performance results presented hereafter remain valid.

→ We propose two novel integrations of PQC into OPC UA's security handshake: *Hybrid OPC UA* and *PQ OPC UA*. The first makes use of hybrid constructions for key exchange, digital signatures, and X.509 certificates. The latter is solely based on post-quantum schemes including PQ X.509 certificates. Both solutions do not alter the existing structure of the security handshake, and our hybrid approach provides backward compatibility to legacy devices. Besides that, we present a novel way for verifying hybrid X.509 certificates using the cryptographic library mbedTLS.

→ We implement and evaluate the two solutions on our ARM-based evaluation platform and provide detailed performance measurements for three NIST security levels. By combining post-quantum key exchange and post-quantum digital signatures we evaluate the total impact of PQC on OPC UA.

→ We analyze the security of our quantum-resistant variants in the symbolic model via the state-of-the-art cryptographic verifier ProVerif. As our integrations target post-quantum confidentiality as well as authentication, we proof both properties in our symbolic models. We construct the formal models of our OPC UA variants in ProVerif's dialect of the applied pi calculus. All formal models presented in this work are available at https://github.com/boschresearch/pq_opc-ua_formal_analysis.

→ Finally, we show that our *PQ* solution outperforms conventional OPC UA in terms of handshake duration at all evaluated security levels. In addition, in four of our six instantiations, we make use of Falcon's highly efficient floating-point implementation, which – to the best of our knowledge – has not been examined in previous performance studies.

1.2 Outline

In Section 2, we introduce the reader to OPC UA and its security mechanisms and, in addition, we provide preliminaries on PQC and formal, computer-aided security analysis. Section 3 highlights related work. In Section 4, we describe our two integrations of PQC into OPC UA. Section 5 presents the symbolic models of our PQ-enabled OPC UA variants. Furthermore, we discuss the results of our formal security analysis. The performance measurements of our two proposed solutions are presented in Section 6. Section 7 concludes our paper.

2 PRELIMINARY BACKGROUND

2.1 OPC UA in Industrial Communication

OPC UA has been specified by the International Electrotechnical Commission (IEC) in the standard series 62541. Furthermore, OPC UA is widely considered a de facto standard for future industrial applications. Because of its service-oriented architecture, OPC UA offers a standardized interface to exchange data between industrial applications independent from manufacturer of automation technology. Recently, it has also been adopted by popular cloud services demonstrating its increasing popularity [9, 54]. OPC UA offers two modes for the transfer of information: a client-server mode and a relatively new publish-subscribe mode [57]. In this work, we focus on the client-server mode since it is widely deployed in current automation systems and fully supported by open-source implementations.

OPC UA provides mutual authentication based on X.509 certificates and it ensures integrity and confidentiality of communication. The bottom layer of OPC UA's security architecture handles the transmission and reception of information. A secure channel is created within the communication layer and is crucial for meeting the aforementioned security objectives. The exchange of information is realized within sessions, which are logical connections between clients and servers.

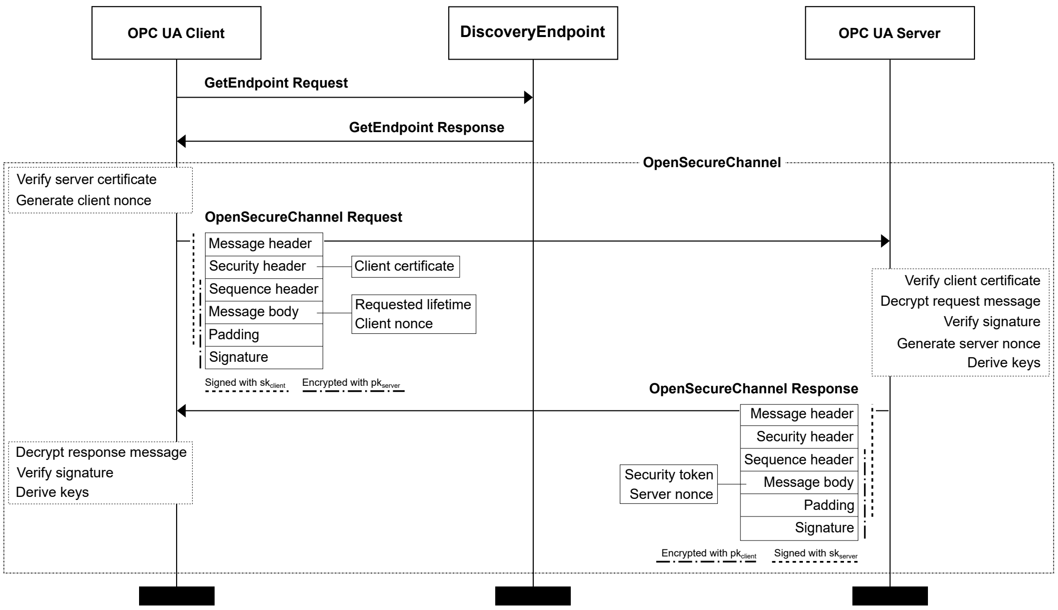


Fig. 1. High-level overview of OPC UA's conventional security handshake.

Figure 1 shows OPC UA's certificate-based authenticated key exchange. The following description of this security handshake is based on the relevant parts of the official specification [58, 59]. Before client and server establish a transport connection, the client issues a *GetEndpoint Request (GetEP Req.)* to the *DiscoveryEndpoint*. In turn, the *DiscoveryEndpoint* sends a *GetEndpoint Response (GetEP Res.)* containing *EndpointDescriptions*, which later allow the client to access services or information offered by the server. In addition, the response contains information required for the security handshake: server certificate, message security mode, and security policy. The server certificate contains the authenticated public key of the server, which the client verifies before initiating the security handshake. OPC UA offers different message security modes for established sessions: *None*, *SignOnly*, and *SignAndEncrypt*. As the name suggests, *SignAndEncrypt* offers confidentiality of communication as well as authenticity, hence, we only consider this security mode in the remainder of this work.

The set of cryptographic mechanisms used during the handshake phase and in subsequent sessions are specified using *SecurityPolicy Profiles*. For example, the security policy *Basic256Sha256* uses RSA2048 to encrypt/decrypt (RSA-OAEP) and sign/verify messages (RSA-PKCS1.5) during the security handshake; symmetric keying material is derived using the hash function SHA256 in a pseudorandom function (PRF); within sessions, AES256 in Cipher Block Chaining mode is used for encryption, and a keyed-hash message authentication code (HMAC) based on SHA256 is used for signatures. In contrast to TLS, OPC UA so far only offers a security handshake that relies on RSA.[2] In essence, it is based on encrypting random client and server nonces that are used to derive session keys.

The following characteristics of the security handshake are specified in the *SecureChannel Service Set*. First, the client sends an *OpenSecureChannel Request (OSC Req.)* to the server. This request

[2]It should be noted that the OPC Foundation plans to standardize a security policy that supports Diffie–Hellman (DH) key exchange based on elliptic curve cryptography (ECC) in the near future [60].

contains a cryptographically secure random number (*client nonce*), a client certificate (including a certificate chain), and a requested lifetime (*RT*) for the secure channel. The request message is encrypted using the authenticated public key of the server and signed using the secret key of the client. In case the verification of the client certificate succeeds, decryption and signature verification take place. Afterwards, the server generates a cryptographic random number (*server nonce*). In order to derive the required session keys, both nonces serve as inputs to a PRF. Two sets of symmetric keys are derived this way: one is associated with the server and the other is associated with the client. The message body of the *OpenSecureChannel Response (OSC Rsp.)* contains a server nonce and a security token (*ST*), the server certificate is placed in the security header of the response message. Secure channels are identified by security tokens, which expire after a specified lifetime. Therefore, the revised lifetime, which is part of the security token, tells the client when to renew the secure channel. The response message itself is encrypted using the client's authenticated public key and signed using the server's private key. After decryption and signature verification, the client derives the keying material from its own nonce and the received server nonce by applying the same PRF as the server. Finally, client and server end up with an identical set of cryptographic keys completing OPC UA's security handshake. The security properties of this handshake have been formally analyzed and the entire security architecture has been investigated in previous works [27, 68].

2.2 Post-Quantum Cryptography

Once a cryptographically relevant quantum computer becomes available, current public key primitives based on the mathematical problem of integer factorization (RSA) and (elliptic curve) discrete logarithm (DH and ECDH) will be broken because of Shor's quantum algorithm [70]. The last decade has seen an increased interest from academia and industry in finding novel cryptosystems that can withstand attacks from quantum computers. In essence, one needs to find a NP-hard problem that is not solvable in polynomial-time by quantum and classical computers.

PQ schemes can be grouped into five families: code-based, lattice-based, hash-based, multivariate, and supersingular EC isogeny cryptography. Out of the five families lattice-based cryptography has arguably attracted the most attention in research: 12 of the remaining 26 schemes in NIST's standardization process are based on lattice problems. Besides that, lattice schemes offer efficient implementations, reasonably sized public keys and ciphertexts, as well as strong security properties [53]. Consequently, we focus on lattice-based cryptography in this work.

A lattice consists of a set of points in a n-dimensional space with a periodic structure. By using n-linearly independent vectors any point in this structure can be reproduced. The security of lattice-based cryptographic primitives are based on NP-hard problems of high-dimensional lattices, such as the shortest vector problem (SVP). All lattice schemes submitted to NIST's standardization process rely on variants of the learning with errors (LWE) problem, learning with rounding (LWR) problem, or NTRU. These problems can be related to aforementioned NP-hard lattice problems via reductions. We investigate the following lattice-based KEMs for potential integration into OPC UA: CRYSTALS-Kyber [8], FrodoKEM [3], LAC [78], NewHope [2], NTRU [31], NTRU-Prime [15], Round5 [10], Saber [34], and ThreeBears [40]. In addition, we investigate the following lattice-based signature schemes: CRYSTALS-Dilithium [37], Falcon [38], and qTESLA [17]. Table 4 and Table 3 in the Appendix list all lattice-based schemes considered in this work including characteristics of their parameter sets.

NIST defined five security levels corresponding to different security strengths in bits for its PQC standardization process. We focus on level 1, 3, and 5 in this work. NIST security level 1 corresponds to 128 bit (classical) security, whereas level 3 and 5 correspond to 192 bit and 256 bit security respectively. KEMs consist of a triple of algorithms: key generation, encapsulation, and decapsulation. Key generation is a probabilistic algorithm that generates a public and private key pair. The probabilistic encapsulation requires a public key as input and generates a shared secret and the corresponding ciphertext. Input to the decapsulation algorithm is a ciphertext and a private key, it either returns a shared secret or an error. Many lattice-based schemes show a small (cryptographically negligible) failure probability during the decapsulation step, in such cases a shared secret cannot be derived. Typically, KEMs offer either indistinguishability under chosen plaintext attack (IND-CPA) or indistinguishability under chosen ciphertext attack (IND-CCA). IND-CPA offers security against passive adversaries, i.e. no information is learned by observing ciphertexts being transmitted. IND-CCA offers a stronger notion of security and provides security in presence of active adversaries. For the integration into OPC UA we rely on an ephemeral key exchange scheme. Any KEM can be easily transformed into an ephemeral key exchange as follows. An initiator generates a public and private key pair and sends its ephemeral public key to a receiving entity. The receiving entity generates a random secret, encrypts it using the received ephemeral public key (encapsulation), and sends the resulting ciphertext back to the initiator. Ultimately, the initiator decrypts the received ciphertext using its ephemeral private key (decapsulation) giving both parties a shared random secret.

Similar to KEMs, signature schemes consist of a triple of algorithms: key generation, signature generation, and signature verification. Key generation returns a public and private key pair. Signature generation takes a private key and a given message to produce a signature. The deterministic signature verification algorithm takes a public key, a message, and a signature and either rejects or accepts the signature. The standard security notion for DSAs is existential unforgeability under chosen message attack (EUF-CMA). NIST required all submitted signature schemes to reach this notion. For specific details of the schemes, we refer the reader to the corresponding specifications.

2.3 Mechanized Security Analysis

Designing cryptographic protocols, such that specific security goals are achieved, is non-trivial and prone to errors, as many past examples have shown [42, 51, 74]. In order to promote trust in cryptographic protocols, their security can be demonstrated via formal analyses or proofs. While it is still common to verify security properties of protocols manually, this approach is no longer recommended by experts for the three following reasons [11]:

(1) Formulating security arguments manually is a complex and time-consuming task. When done on paper, errors tend to go unnoticed.
(2) To make manual analysis feasible protocols are represented as simplified models. In case of over-simplifications, design flaws could be missed.
(3) Over the last decade (semi-)automated verification tools have matured, resulting in trustworthy formal analyses of widely deployed protocols [14, 41, 47, 50, 79].

In the realm of computer-aided verification, two strains of mathematical models have solidified: security analysis in the *symbolic model* and in the *computational model*. While computational models are more realistic, they are burdensome and error-prone to model – especially when considering complex protocols. Besides that, proofs in the computational model are only semi-automatic, i.e. require user input. The symbolic model, on the other hand, allows for complete automation, even with complex protocols [11]. As a result, we focus on tool-based verification in the symbolic model in this work. For the sake of completeness, we also briefly discuss the computational model in the following.

2.3.1 Computational Model. In the computational model, adversaries are represented as probabilistic Turing machines, messages are modeled as bit strings and cryptographic primitives as probabilistic functions from bit strings to bit strings [22]. Computer-aided proofs in the computational model are typically game-based.[3] In fact, a proof methodology called game-hopping is applied: the goal is to demonstrate that breaking a specific security property is possible only with negligible probability. Starting at the original game with unknown success probability, sequential transformations allow to reach games that enable the computation of the adversary's success probability. Via reductions to known complexity assumptions, e.g. discrete logarithm problem, it is possible to infer the adversary's success probability for the original game.

Compared to symbolic models, proofs based on computational models yield stronger guarantees. However, formal analyses in the computational model are very difficult to mechanize; so far, only semi-automated tools exist.

CryptoVerif. The first tool to tackle mechanized verification of computational models was CryptoVerif [24]. During the formal analysis, it generates intermediate games automatically with little to no user interaction – depending on protocol complexity. CryptoVerif is capable of proving confidentiality and authentication properties. As its input language is based on applied pi calculus, it is especially well suited for protocol analysis.

2.3.2 Symbolic Model. The symbolic model is a simpler, abstract model and is due to Dolev and Yao [36] as well as Needham and Schroeder [56]. Messages are modeled as terms that cannot be split into compound bit strings. Cryptographic primitives are represented by black-box functions that operate on these terms. Furthermore, perfect cryptography is assumed. This implies the following [41]:

- Encrypted messages reveal nothing about the plaintext.
- Signatures are unforgeable.
- Hash functions are, in essence, random oracles with no collisions.
- Random numbers are truly random with no repetitions.

For instance, to break confidentiality properties the adversary needs to be in possession of the secret decryption key. Furthermore, it is possible to add cryptographic primitives by defining them as new rewrite rules or equations. For example, we formally model KEMs in our PQC-enabled OPC UA variants using function symbols and rewrite rules (see Section 5). Note that for all added (post-quantum) cryptographic constructions the perfect cryptography assumption applies as well.

The attack model considers the classical Dolev–Yao model [36]. Consequently, the adversary has complete control over the network, eliminating the need to model dishonest parties [22, 25]. In essence, the Dolev–Yao model enables an adversary to read, remove, replay, and modify messages at will. However, the computational capabilities of this powerful adversary are restricted to the defined primitives.

Two classes of symbolic security properties exist: trace properties and equivalence properties. On the one hand, trace properties state that a specific property holds on every protocol run. Confidentiality may be defined based on trace properties, meaning an adversary does not obtain knowledge of certain data, such as secret keys. Authentication is typically expressed using correspondence assertions, which is a subclass of trace properties. Equivalence properties, on the other hand, express that an adversary is not capable of differentiating between two protocols. One intuitive example of an equivalence property is that an adversary cannot distinguish between a protocol containing the real secret or a random value [11].

While it is easier to automate formal protocol verification in the symbolic model than in the computational model, certain challenges remain – mainly because of the infinite state space to

[3] Any security game consists of an adversary and a challenger. In case the adversary achieves a goal condition, it wins the game, i.e. breaks the scheme.

explore [22]. Furthermore, the absence of attacks in the symbolic model does not generally prove it secure in the computational model. However, security analyses of protocols in the symbolic model have been highly appreciated, especially in the standardization of new protocols (see Section 3.2). Due to this mechanized approach protocol changes are also easier to regulate than in the pen-and-paper approach. This makes tool-based verification in the symbolic model a valuable approach for detecting logical flaws in protocols.

Over the past years several symbolic verification tools have been introduced: *CPSA*, *F7*, *Maude-NPA*, *ProVerif*, *Scyther*, *Tamarin*, and *Verifpal* [11, 48]. In the domain of automated symbolic verification tools, Tamarin and ProVerif stand out and are considered state-of-the-art tools. In fact, they both have been used to evaluate large-scale, real-world protocols like TLS 1.3 and Noise. As we build on the already existing symbolic proof for OPC UA in [68], which is based on ProVerif, we use ProVerif for our proofs of the PQC-enabled OPC UA variants.

ProVerif. As mentioned above, ProVerif stands out in the realm of symbolic verification tools [21–23, 25]. In fact, many real-world protocols have been verified using ProVerif: Signal [47], Noise [45], TLS 1.3 [16], and others [14, 50, 79]. As is common for symbolic models, ProVerif allows to verify various trace and equivalence properties. In addition, it analyzes protocols with respect to an unbounded message space and an unbounded number of sessions.

Protocols are modeled using a variant of the applied pi calculus language [1]. ProVerif then translates the modeled protocol into a set of Horn clauses, which it automatically verifies. For the verification process, the security properties also need to be translated into derivable queries on the resulting Horn clause representation. In case ProVerif does not find an attack, the desired property is proven secure. Moreover, ProVerif has also been proven to not miss any attacks [48]. Note that false attacks may be found, especially when modeling protocols with global states [22, 50]. And since ProVerif does not bound the number of executed protocol sessions and message space, it does not always terminate. Apart from these two minor drawbacks, it has proven to be of great value in the verification of cryptographic protocols.

3 RELATED WORK

3.1 Integration of PQC into Protocols

There have been a lot of research efforts integrating PQC into widespread Internet protocols such as TLS, SSH (Secure Shell), and IKEv2 (Internet Key Exchange version 2). Since OPC UA's security handshake is loosely inspired by TLS's handshake protocol, we focus on previous works in this area. In general, existing integration studies can be grouped into the following three categories: standardization efforts, implementation works, and experimental studies. Two active Internet Engineering Task Force (IETF) Internet-Drafts exist that describe the integration of hybrid key exchange into TLS 1.2 [30] and TLS 1.3 [72]. Many experimental studies have been conducted under real network conditions [26, 49, 71] or under lab conditions [33, 62]. In aforementioned studies, the authors typically make use of already existing open source implementations of PQC. For example, Open Quantum Safe provides prototypical integrations of PQ schemes into the popular library OpenSSL [73]. Other works exist where PQC has been either integrated into embedded libraries [29] or has been optimized for specific platforms [44]. Our implementations of PQ schemes are mainly based on PQClean,[4] which provides portable implementations for an easy integration into other codebases. When investigating authentication, another difficulty must be dealt with: a long-term public key is involved, which is typically stored and distributed via certificates. Previous works proposed hybrid certificates for the post-quantum transition where extension fields are used to bind

[4]https://github.com/PQClean/PQClean.

an additional public key to an entity using an additional PQ signature scheme [18, 20]. In addition, the impact of hybrid and PQ certificates on various Internet protocols has been investigated [43, 71].

Since it enables confidentiality against future quantum adversaries, hybrid key exchange has so far attracted the most attention. If authentication and key exchange are considered, they are typically evaluated separately, hence not showing the entire impact of PQC. Hybrid authentication has been addressed, but it was evaluated separately from key exchange and no performance measurements were conducted [33]. The authors of [29] investigated the combined impact of PQ key exchange and authentication on TLS for embedded devices, but only considered one set of PQ primitives at one security level.

3.2 Automated Security Analysis of Cryptographic Protocols

The first automated verification tools arrived roughly two decades ago [55, 76]. While these tools were considered impressive advances in the academic community, they were not suitable for analyzing complex real-world protocols. They often lacked completeness, did not guarantee termination, and, in addition, could not prove the absence of attacks since they only analyzed a subset of the state space.

Eventually, powerful verification tools, such as ProVerif [21, 23, 32], emerged. Over they years, ProVerif has amassed a remarkable track record in verifying and detecting flaws in numerous protocols [14, 47, 50, 79]. In fact, it even played a major role in the standardization process of TLS 1.3 [16, 45]. With the introduction of tools that promise high accessibility and usability, e.g. Verifpal [48], tool-based verification of protocols may be on the verge of broader adoption in the realm of practitioners.

OPC UA's security handshake has been verified in the symbolic model using ProVerif [68]. This analysis focused on confidentiality and authentication properties. Attacks were found that affected authentication properties within the security modes *Sign* and *SignAndEncrypt*. The authors of [68] provided countermeasures that have been communicated and clarified with the OPC Foundation [67]. In essence, the public key of the receiving entity (in form of a hash of the certificate) must be included in the *OpenSecureChannel Request* and *Response* otherwise man-in-the-middle attacks are possible. Note that we base the ProVerif models of our proposed PQ variants on the fixed model presented in [68].

In further related works, the integration of PQC into other protocols have been formally verified in the symbolic model. For instance, in [41], the authors introduce a post-quantum variant of the WireGuard VPN protocol, which they formally verify using the Tamarin protocol prover [13].

4 INTEGRATION OF PQC INTO OPC UA

4.1 Hybrid OPC UA

In hybrid modes, different options for combining cryptographic material exist. We use the XOR-then-MAC combiner from [19] regarding confidentiality of data, which is provably secure against fully quantum adversaries. Besides the integration of a hybrid key exchange scheme, we need to convey two long-term public keys and two digital signatures for authenticity and integrity. For reasons of backward compatibility, we work with X.509 certificates that consist of two non-critical extensions as proposed in [18]. The first contains the public key of the additional PQ signature scheme, the second holds the signature over the certified data. Messages are signed independently from each other using two different signature schemes. The security properties of this concatenation combiner have been estblished in [20]. While the merits of a hybrid key exchange are obvious, there is a slightly weaker need for hybrid authentication and hybrid digital signatures. However, applications will have to support conventional and PQ schemes in order to be backward compatible

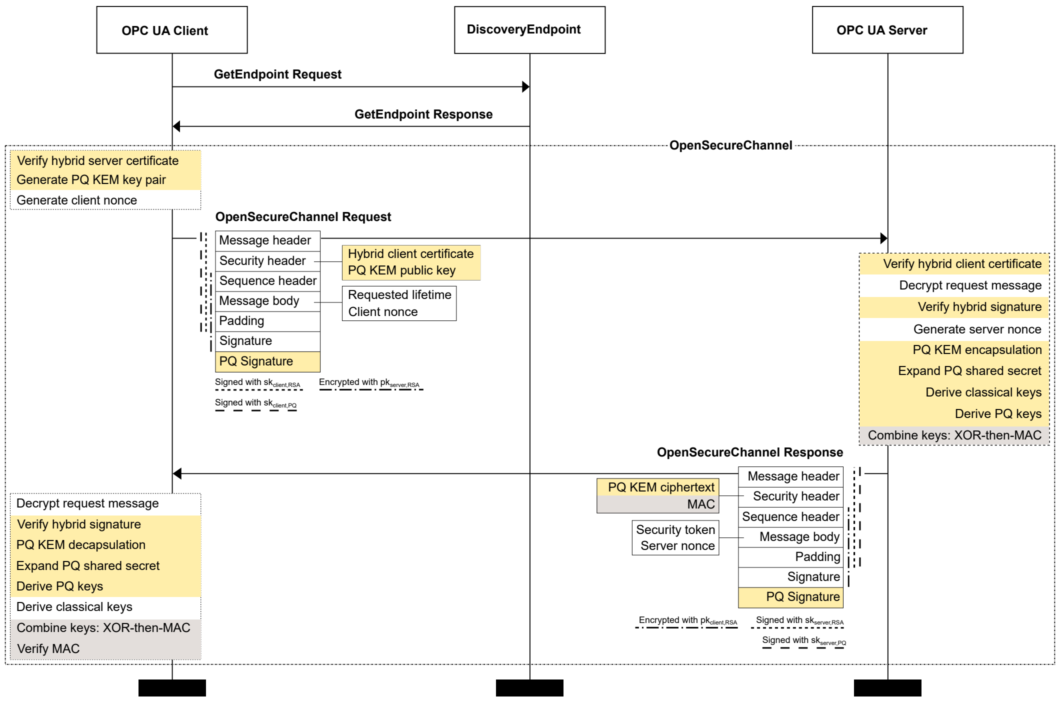


Fig. 2. High-level overview of Hybrid OPC UA (yellow: operations/data related to PQC; gray: operations/data related to XOR-then-MAC combiner).

with applications, which have not been upgraded yet. Therefore, we also consider hybrid signatures and authentication in this work to fully understand its impact on OPC UA.

The integration of hybrid modes into the security handshake of OPC UA requires modifications to the *SecureChannel Service Set*. Figure 2 gives an overview of our hybrid integration of PQC into OPC UA's security handshake; modifications related to PQC are marked in yellow, while changes related to the XOR-then-MAC construction are marked in gray. At first, we define a new security policy *Hybrid{1,3,5}_Basic256* that the server suggests to the client within the *GetEndpoints Response*. In our approach, this response contains the hybrid X.509 certificate (including the certificate chain). First, the client verifies the entire certificate chain assuming a hybrid root certificate has been preinstalled. In addition to a random client nonce, the ephemeral key generation function of a PQ KEM needs to be called (pk_{PQ}, sk_{PQ}). The hybrid *OSC Req.* is initialized using the client nonce, pk_{PQ}, and the security settings obtained from the *GetEndpoints Response*. The additional public key is positioned within the security header, which also includes the hybrid client certificate. Before the request is sent to the server in form of an OPC UA message, it is signed using the aforementioned hybrid signature scheme: A hash is computed over the entire message that is then signed conventionally and by a PQ signature scheme. According to the specification of OPC UA, the sequence header, the message body containing the client nonce, and the message footer containing RSA-padding fields and signatures are encrypted. We avoid expensive RSA encryption/decryption by placing the additional values of our hybrid solution (pk_{PQ} and PQ signature) outside the encrypted message parts.

Once the server receives the request, it verifies the hybrid client certificate (including the certificate chain). After the certificate verification, the conventionally encrypted message parts are

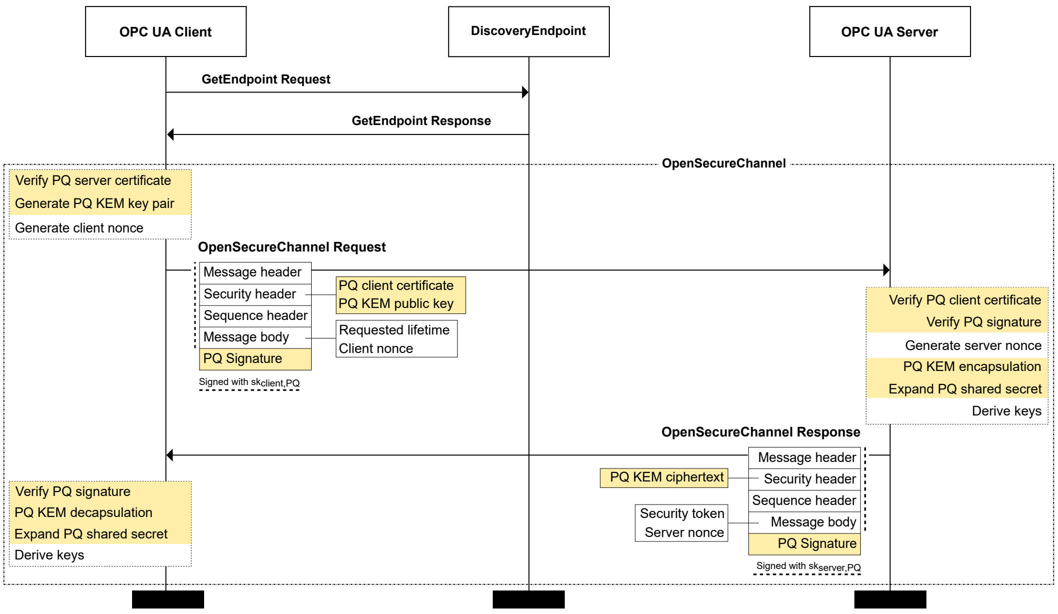


Fig. 3. High-level overview of PQ OPC UA (yellow: operations/data related to PQC).

decrypted and the two signatures are verified. As in conventional OPC UA, the server then creates his server nonce. For our proposed hybrid mode, the encapsulation function of the respective PQ KEM is called using the received public key pk_{PQ} as input. This generates a ciphertext ct_{PQ} and a shared secret ss_{PQ}. In order to maintain the original structure of OPC UA's security handshake, we expand the shared secret using a PRF to obtain additional nonce values. Further calls to PRFs generate two types of keying material: a conventional set and a post-quantum set. In a subsequent step, the two sets are combined using XOR. To complete the XOR-then-MAC combiner, we compute a MAC over the ciphertext ct_{PQ} and the original server and client nonce using the generated server's symmetric signing key. The ciphertext and MAC are placed in the security header. We keep the server nonce inside the body of the response message alongside the revised lifetime of the secure channel. The response message is signed using the aforementioned concatenation combiner. After signing the message, the sequence header, the message body, and message footer are encrypted. Again, this avoids expensive encryption of the additional, potentially large cryptographic material: post-quantum ciphertext ct_{PQ} and signature.

The client receives the response, conventionally decrypts it, and verifies the included hybrid signature. Utilizing the received PQ ciphertext ct_{PQ} and the client's own PQ KEM secret key sk_{PQ}, the corresponding decapsulation function of the respective KEM is called, which outputs the shared secret ss_{PQ}. As in processing the *OSC Req.*, this shared secret is expanded to create additional nonce values. Having obtained all required nonces, we generate two types of keying material (conventional and PQ) and combine them using XOR. We verify the received MAC by using the computed symmetric signing key completing our hybrid security handshake.

4.2 Post-Quantum OPC UA

Once PQ schemes have been standardized, they will be adopted in protocols and will be considered state-of-the-art. Consequently, hybrid modes will not be required any longer. For our *PQ OPC UA* solution, we keep the structure of the original security handshake but replace conventional asymmetric primitives with PQ key encapsulation and digital signature schemes. Figure 3 provides an overview of the modified security handshake only based on PQC primitives; modifications are marked in yellow.

We introduce a new security policy *PQ{1,3,5}* that is sent to the client in *GetEndpoints Response*. The conveyed server certificate contains a single PQ public key and is signed with a PQ signature scheme. The client verifies the server certificate including the certificate chain. Again, we assume the PQ root certificate has been preinstalled on both client and server. The generation of the *OSC Req.* is the same as in our hybrid mode. First, a random client nonce is created and then the ephemeral key pair of a PQ KEM (pk_{PQ}, sk_{PQ}). Since we base the key exchange of our PQ solution solely on a PQ KEM, we do not require secrecy of the random client and server nonce. As a consequence, sequence header, message body, and message footer of the *OSC Req.* and *OSC Rsp.* are sent unencrypted. The resulting *OSC Req.* is signed using the client's private PQ signing key, the certificate containing the corresponding PQ public key is part of the request message sent to the server.

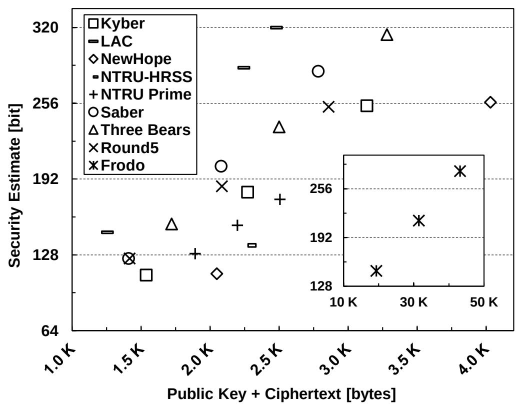
The server verifies the PQ client certificate (including the certificate chain) and the signature of the *OSC Req.* using the client's authenticated public key. After the verification step, the encapsulation function of the KEM is invoked resulting in a ciphertext (ct_{PQ}) and shared secret (ss_{PQ}). Besides that, we generate a random server nonce. The shared secret and both random nonces serve as input to a PRF. We consider the output of the PRF our *master secret*. Subsequently, we use the *master secret* as input to another PRF to obtain symmetric keying material. By keeping the random nonces from the conventional security handshake and by using them as input to the first PRF we ensure that both parties contribute to the *master secret*. The *OSC Rsp.* contains the generated ciphertext, the server certificate, the server nonce, and the revised lifetime of the secure channel. The response is signed using the server's private PQ signing key, and the signature is appended to the response message.

Once the client receives the *OSC Rsp.*, the signature is verified using the server's authenticated public key. Then, the client calls the decapsulation function of the PQ KEM resulting in the shared secret (ss_{PQ}). Again, this shared secret serves as input to a PRF alongside the client and server nonce. The output is fed to another PRF to compute the final keying material. Server and client derive the same keying material, which is used in subsequent communication sessions. This completes OPC UA's handshake solely based on PQ schemes: Client and server are mutually authenticated via PQ certificates and signatures; Keying material is derived using a key exchange scheme based on a PQ KEM.

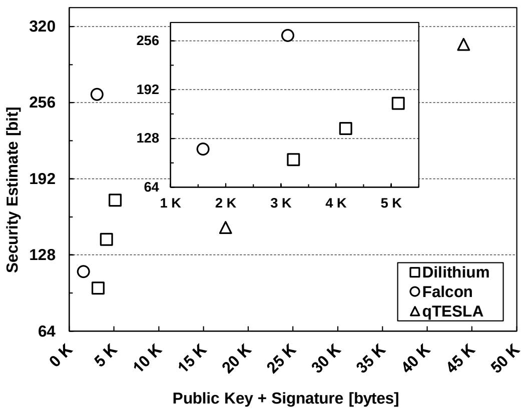
4.3 Selection of Quantum-Resistant Primitives

In principle, our generic approach allows us to integrate any KEM and DSA. Our criteria for the selection of PQC algorithms are as follows. We require lattice-based schemes that offer a balanced trade-off in terms of estimated security, public key + ciphertext/signature size, and performance since the time to establish a secure channel should not substantially increase. In addition, we only consider algorithms that are part of NIST's ongoing PQC standardization process (Round 2). Consequently, their official specification should offer various parameter sets that cover different security levels; KEMs should provide IND-CCA. Integration into OPC UA needs to be possible without any modifications to cryptographic algorithms since we do not want to invalidate any of their security claims.

4.3.1 Security-Size Trade-Off. First, we study the trade-off in terms of security and size of all remaining lattice-based Round 2 submissions. The size metric is important to allow for an easy integration into existing protocols. In our case, the size metric for KEMs consists of the public



(a) Key encapsulation mechanisms.



(b) Digital signature algorithms.

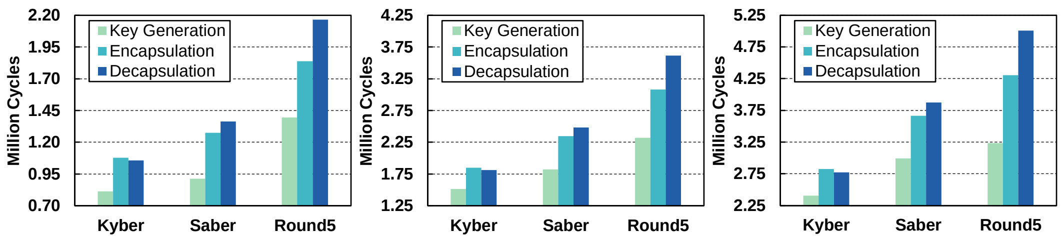
Fig. 4. Security-size trade-off for lattice-based quantum-resistant schemes.

key and ciphertext size since both need to be transmitted in our proposed solutions. Regarding DSAs, we use public key and signature size as metric. Both are transmitted via certificates to other nodes during the handshake. Considering the security metric, we use security strength estimations provided in the specification of each submission. These figures are based on the estimated cost of the best known attacks against the underlying lattice-problem, typically core-SVP hardness is evaluated.

Figure 4 shows the trade-off for estimated security and size for lattice-based schemes remaining in NIST's PQC process. Note that for submissions containing multiple schemes or multiple parameter sets, we only consider one scheme or one set of parameters. In case of NTRU, we consider the recommended KEM parameter set NTRU-HRSS; for NTRU Prime, we only consider the parameter sets of Streamlined NTRU Prime. For Round5, which specifies a total of 21 parameter sets, we only consider their specified IND-CCA secure KEM with ring parameter set and no error correction, i.e. R5ND_CCA_0d_KEM.

Our evaluation shows that parameter sets for Kyber (Kyber512, Kyber768, and Kyber1024), Round5 (R5ND_1CCA_0d, R5ND_3CCA_0d, and R5ND_5CCA_0d), and Saber (LightSaber, Saber, and FireSaber) offer a very good trade-off in terms of public key + ciphertext size and estimated security strength. Consequently, we select these three schemes for a further performance evaluation. From the trade-off in Fig. 4(a), LAC seems like another promising candidate. However, attacks on LAC that allow to fully recover the secret key have been discovered decreasing our trust in this scheme [35, 39]. We do not select other schemes for further evaluation because (a) their parameter sets imply an imbalanced security-size trade-off (NTRU-HRSS, NewHope, and Frodo), (b) they have not attracted much attention in previous experimental studies (Three Bears and NTRU Prime), or (c) known attacks significantly reduce their security estimations (LAC).

The security-size trade-off for digital signature schemes is shown in Fig. 4(b). After an update to its Round 2 specification, qTESLA only provides provably-secure parameter sets that come with very large sizes for signatures and public keys. Ultimately, we select the remaining two signature algorithms – Falcon and Dilithium – for a further performance evaluation. Both seem to be promising signature algorithms since public key and signature are reasonably sized and they provide parameter sets for different security strengths (level 1: Falcon512 and Dilithium2, level 3: Dilithium4, level 5: Falcon1024).



(a) Level 1 parameter sets. (b) Level 3 parameter sets. (c) Level 5 parameter sets.

Fig. 5. Average performance of selected key encapsulation mechanisms.

4.3.2 Preliminary performance evaluation. We continue with an evaluation of the standalone performance of the selected algorithms on our target platform – Raspberry Pi 3 Model B (see Section 6.2). In order to obtain cycle-accurate measurements, we added a kernel extension that enables access to the CPU cycle count register [5]. Our goal is to select parameter sets for three security levels with a balanced trade-off in terms of security, size, and performance. Our implementations of Kyber and Saber are based on code from PQClean. Round5 has not been integrated there; consequently, we work with code from the official Round5 submission package.[5] Figure 5 shows the average cycle counts of 100 executions of the selected KEMs. Across all security levels Kyber shows the best performance. Considering all processing steps of KEMs, Kyber is significantly faster than Round5 (in average 3.6×10^6 cycles at each security level) and also faster than Saber (in average 1.5×10^6 cycles at each security level). In comparison, the standalone performance of an ECDHE key exchange based on SECP256R1, which corresponds to security level 1, takes 2.1×10^7 cycles on our evaluation platform, whereas Kyber512 only takes 2.9×10^6 cycles. Kyber has also been part of several previous studies resulting in a similar assessment of its performance [29, 62]. Consequently, we select the three parameter sets of Kyber for instantiating our solutions.

Having analyzed KEMs, we turn to the two selected signature schemes. Exploiting Falcon's floating-point arithmetic requires an underlying hardware floating-point unit (FPU) to support double-precision floating-point as defined by the IEEE 754 standard [65]. For devices without hardware FPU, an implementation exists that emulates floating-point precision (Falcon-EMU). The ARMv8 instruction set of the Raspberry Pi 3 fulfills the aforementioned requirement, which allows us to evaluate both implementations, i.e. Falcon-FPU and Falcon-EMU [6]. Our implementation of Dilithium is based on code from PQClean. For the implementation of Falcon, we make use of reference code from the official website.[6] Figure 6 shows the average cycle counts of signature generation and verification of the selected DSAs in comparison with ECDSA and RSA over 100 executions. Please note, we do not report performance measurements of key generation since generation of new signing keys is typically required only rarely. Enabling floating-point operations by using Falcon-FPU increases signature generation in average 11.4 times compared to Falcon-EMU. Furthermore, Falcon's highest security parameter set is even 1.9×10^6 cycles faster than Dilithium's level 1 configuration in case floating-point operations are enabled. All parameter sets of Dilithium and Falcon-FPU outperform the conventional signature scheme ECDSA based on the elliptic-curve SECP256R1, which corresponds to security level 1. The total runtime (signature generation plus verification) of SECP256R1 corresponds to 3.2×10^7 cycles on our evaluation platform. In comparison, Falcon512-FPU only takes 4.7×10^6 cycles and Dilithium2 1.1×10^7 cycles. Since Falcon provides very efficient sizes for signatures and public key and since our evaluation platform is able to use

[5]https://github.com/round5/code/tree/master/configurable.
[6]https://falcon-sign.info.

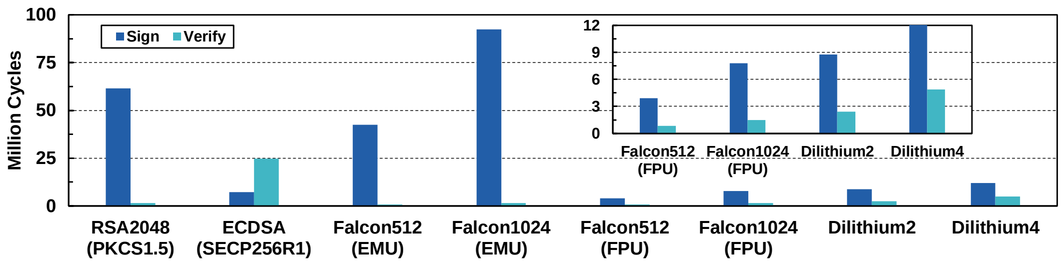


Fig. 6. Average performance of selected digital signature algorithms.

Falcon's floating-point arithmetic, we select it for instantiating our proposed solutions. However, Falcon does not offer a parameter set covering security level 3. Therefore, we use Dilithium4 within the instantiation targeting that security strength. Besides that, we are not aware of any works that have shown fundamental weaknesses in either Falcon or Dilithium, and both have been part of previous experimental studies [62, 71].

In accordance with our initial requirements, we instantiate our two proposed solutions with the following algorithms: We use Kyber512 and Falcon512-FPU regarding NIST security level 1, for security level 3 we use Kyber768 and Dilithium4, and for level 5 we work with Kyber1024 and Falcon1024-FPU.

5 MECHANIZED SECURITY ANALYSIS OF PROPOSED INTEGRATIONS

Next, we present our security analysis in the symbolic model of the proposed OPC UA variants. First, we revisit the formal model of conventional OPC UA presented in [68], which forms the basis of all our ProVerif models.[7] Second, we present the formal models of our two variants: Hybrid and PQ OPC UA. As we rely on the symbolic model for our proofs, we assume perfect cryptography and the Dolev–Yao attack model (see Section 2.3 for details). Besides that, we only consider the security mode *SignAndEncrypt* since it is the only setting offering confidentiality of communication as well as mutual authentication between client and server.

We use the following notation in our subsequent graphical depictions of our models:

- $\mathsf{asym\_enc}(\mathsf{m};\mathsf{pk}_\mathsf{X})$ and $\mathsf{asym\_dec}(\mathsf{c};\mathsf{sk}_\mathsf{X})$: The former denotes that a message m is asymmetrically encrypted under the public key pk_X, while the latter denotes the decryption of the ciphertext c under the corresponding secret key sk_X.
- $\mathsf{sign}(\mathsf{m};\mathsf{sk}_\mathsf{X})$ and $\mathsf{verify}(\mathsf{sig},\mathsf{m};\mathsf{pk}_\mathsf{X})$: A message m is signed using the secret key sk_X and the resulting signature can be verified using the corresponding public key pk_X. In case of our proposed variants, the index X also states whether a classical (RSA) or post-quantum public key primitive (PQ) is used.
- $\mathsf{kem\_keygen}()$, $\mathsf{kem\_encap}(\mathsf{pk}_\mathsf{PQ})$, and $\mathsf{kem\_decap}(\mathsf{ct}_\mathsf{PQ};\mathsf{sk}_\mathsf{PQ})$: Our integrations of PQC into OPC UA require KEM constructions for establishing a shared secret. A key pair (sk_PQ, pk_PQ) is first generated using the KEM's key generation function. The encapsulation operation takes as input a KEM public key pk_PQ and outputs a shared secret ss_PQ and the corresponding ciphertext ct_PQ. The shared secret can be obtained by the entity holding the respective KEM secret key via the decapsulation function, which takes as input the ciphertext ct_PQ and the secret key sk_PQ.
- $\mathsf{h}(\mathsf{m})$: Applies a hash function h to the message m.

[7] We used ProVerif version 2.02pl1 for our proofs presented in this work.

Listing 1. Main process of the conventional OPC UA model in ProVerif.

```
process
(* Construct key pairs and output public keys of
all involved parties on the public communication channel *)
let pkC = pk(skC) in out(c, pkC);
let pkS = pk(skS) in out(c, pkS);
let pkEP = pk(skEP) in out(c, pkEP);

(* Start protocol *)
(
!process_client(skC, skEP, skS) |
!process_server(skC, skEP, skS) |
!process_endpoint(skC, skEP, skS)
)
```

Listing 2. Analyzed security properties in our OPC UA models.

```
(* Queries *)
query attacker(messageCli_S(pk(skC), pk(skS))).
query attacker(messageSrv_S(pk(skC), pk(skS))).
query attacker(messageCli_C(pk(skS), pk(skC))).
query attacker(messageSrv_C(pk(skS), pk(skC))).

(* Correspondence Assertions *)
query X: host, Y: host, N: nonce ; event(terminateOSC_Server(X, Y, N)) ==>
    event(initiateOSC_Client(X, Y, N)).
query X: host, Y: host, N: nonce ; event(terminateOSC_Client(X, Y, N)) ==>
    event(initiateOSC_Server(X, Y, N)).
```

– PRF(x, s): The pseudorandom function PRF takes as input the variable x and a secret seed value s.
– MAC(m; k) and verify_MAC(mac, m; k): As our hybrid variant relies on the XOR-then-MAC combiner, we use a MAC function, which takes as input a message m and a MAC key k, to generate a MAC-tag. To verify the MAC-tag we use the function verify_MAC, which takes as input the received tag, a message m, and the corresponding MAC key k.

As the symbolic proofs of our proposed OPC UA variants build on the existing proof in [68], we rely on their assumptions in our ProVerif models. Besides that, we limit our analysis to the security mode *SignAndEncrypt* and modify their original model to reflect OPC UA's most current specification. As a result, we assume that clients always accept the proposed security mode *SignAndEncrypt*, which is part of the *GetEndpoint Response* sent by the *DiscoveryEndpoint*. In industrial networks, this can be achieved via administrative policies and by restricting clients and servers to only support the security mode *SignAndEncrypt*, which offers the most security guarantees, i.e. confidentiality and authentication. As in [68], we model certificates as public keys and, thus, consider the complexities of an underlying public key infrastructure (PKI) and its respective operations out of scope, e.g. certificate verification, renewal, and revocation. We expect this assumption not to exclude attacks on the desired authentication properties since in highly regulated industrial control systems certificate management is typically achieved via out-of-band mechanisms. In our models, we place the public keys of the three involved parties (OPC UA Client, DiscoveryEndpoint, OPC UA Server) on the communication channel in the main process of our models, which are then implicitly trusted by the different entities. For instance, this is highlighted in Listing 1, which shows the main process of the conventional OPC UA model written in ProVerif's variant of the applied pi calculus language: the public keys are placed on the communication channel first and then OPC UA's security handshake is initiated.

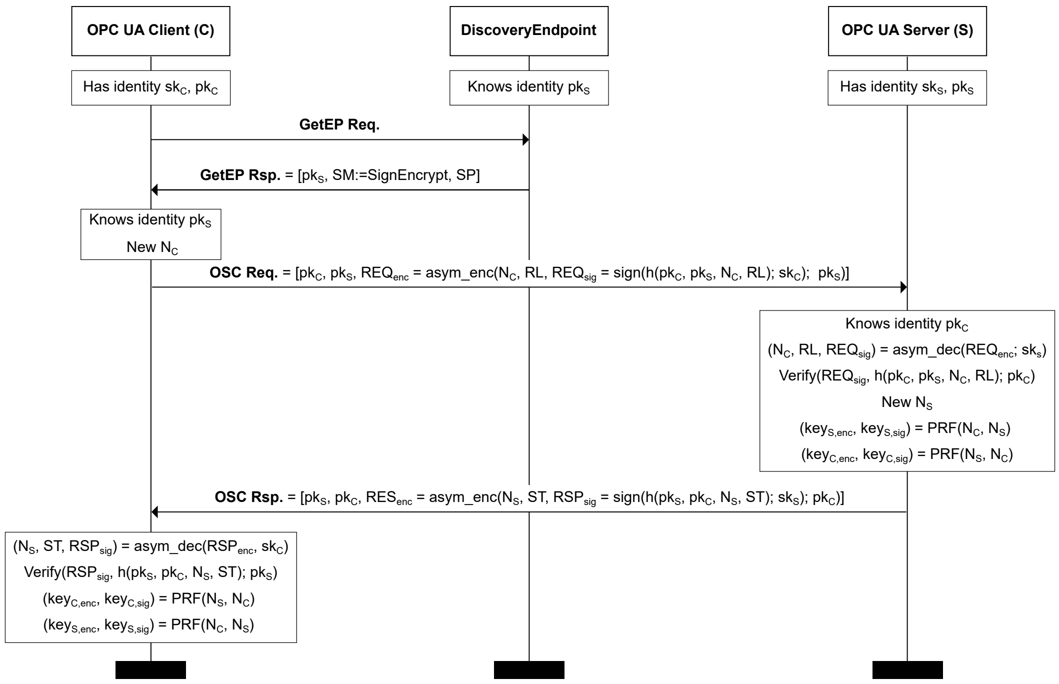

OPC UA Client (C)
DiscoveryEndpoint
OPC UA Server (S)
Has identity sk_C, pk_C
Knows identity pk_S
Has identity sk_S, pk_S
GetEP Req.
GetEP Rsp. = [pk_S, SM:=SignEncrypt, SP]
Knows identity pk_S
New N_C
OSC Req. = [pk_C, pk_S, REQ_{enc} = asym_enc(N_C, RL, REQ_{sig} = sign(h(pk_C, pk_S, N_C, RL); sk_C); pk_S)]
Knows identity pk_C
(N_C, RL, REQ_{sig}) = asym_dec(REQ_{enc}; sk_s)
Verify(REQ_{sig}, h(pk_C, pk_S, N_C, RL); pk_C)
New N_S
($key_{S,enc}$, $key_{S,sig}$) = PRF(N_C, N_S)
($key_{C,enc}$, $key_{C,sig}$) = PRF(N_S, N_C)
OSC Rsp. = [pk_S, pk_C, RES_{enc} = asym_enc(N_S, ST, RSP_{sig} = sign(h(pk_S, pk_C, N_S, ST); sk_S); pk_C)]
(N_S, ST, RSP_{sig}) = asym_dec(RSP_{enc}, sk_C)
Verify(RSP_{sig}, h(pk_S, pk_C, N_S, ST); pk_S)
($key_{C,enc}$, $key_{C,sig}$) = PRF(N_S, N_C)
($key_{S,enc}$, $key_{S,sig}$) = PRF(N_C, N_S)

Fig. 7. Formal model of conventional OPC UA in security mode SignAndEncrypt.

Regarding security objectives, we consider confidentiality as well as authentication properties in our models. Confidentiality of the resulting client and server keying material is checked using *queries*. By testing the resulting client and server keying material individually we are able to examine their confidentiality properties separately in case of any failures. Authentication properties are analyzed using *correspondence assertions*. In essence, correspondence assertions allow to study the relationship between *events* that resemble important protocol stages but do not affect the overall protocol flow. In case of our OPC UA models, we place events at the beginning and the end of the respective client and server processes. By doing so we prove the following properties: Whenever the server S reaches the end of OPC UA's security handshake it believes it has done so with client C – and vice versa.

We query confidentiality and authentication properties in similar fashion in all proposed models. Listing 2 shows the defined queries and correspondence assertions in case of the conventional model. In case of our PQC-enabled OPC UA variants, they are modified to account for the KEM-based keying material.

5.1 Symbolic Proof: Conventional OPC UA

As mentioned above, Puys et al. [68] already analyzed OPC UA in the symbolic model using ProVerif. Based on this analysis, they identified a weakness that enabled an attacker to mount a man-in-the-middle attack on OPC UA's security handshake. Their proposed mitigation fixes this attack.

While examining the fixed ProVerif model of Puys et al., we found that their modeled request and response messages do not completely adhere to OPC UA's message structure. In OPC UA, only the sequence header, the message body (incl. padding), and the signature are encrypted. Whereas the signature is based on all parts of the message. Hence, two header fields are only signed but

Listing 3. Modeling OPC UA's message structure in ProVerif.

```
(* send OpenSecureChannelRequest *)
new RL: bitstring;
let sig_OPN_Request = sign(skClient, hash_ideal((OpenSecureChannelRequest,
pkClient, Server, nonce_to_bitstring(Nclient), RL))) in
let enc_OPN_Request = rsa_enc(pkServer, (nonce_to_bitstring(Nclient), RL,
sig_OPN_Request)) in
event initiateOSC_Client(Client, Server, Nclient);
out(c, (OpenSecureChannelRequest, pkClient, Server, enc_OPN_Request));
```

Table 1. ProVerif's verification results of our analyzed OPC UA models.

Security Objective	*Entity*	*Query/Correspondence Assertion*	*Conventional*	*Hybrid*	*PQ*
Confidentiality	OPC UA Client	Client Encrypting Key $(\text{key}_{\text{C,enc}})$	True	True	True
		Server Encrypting Key $(\text{key}_{\text{S,enc}})$	True	True	True
		Client Signing Key $(\text{key}_{\text{C,sig}})$	True	True	True
		Server Signing Key $(\text{key}_{\text{S,sig}})$	True	True	True
	OPC UA Server	Server Encrypting Key $(\text{key}_{\text{S,enc}})$	True	True	True
		Client Encrypting Key $(\text{key}_{\text{C,enc}})$	True	True	True
		Server Signing Key $(\text{key}_{\text{S,sig}})$	True	True	True
		Client Signing Key $(\text{key}_{\text{C,sig}})$	True	True	True
Authentication	OPC UA Client	Client $\longrightarrow$ Server	True	True	True
	OPC UA Server	Server $\longrightarrow$ Client	True	True	True

not encrypted: message and security header, which include the message type, the certificate of the sending entity, and a hash over the certificate of the receiving entity. To demonstrate this behavior we depict the part of our modeled client process in Listing 3, where the client sends its request message to the server. Note that `OpenSecureChannelRequest`, as part of the message header, as well as `pkClient` and `Server`, as part of the security header, are sent unencrypted.

In order to ensure conformance with the official specification of the OpenSecureChannel Service [59], we apply these modifications to our model. Figure 7 shows the resulting ProVerif model of the conventional OPC UA handshake.

As a result, ProVerif does not find any attacks on OPC UA's conventional handshake. In fact, it proves all of the confidentiality and authentication queries; Table 1 summarizes the verification results of our analysis. Considering verification runtime, our conventional OPC UA model finishes in less than one second wall time (0.527 s) on a notebook running Ubuntu 20.04.02 LTS with an Intel Core i7-8650 CPU clocked at 2.11 GHz and 16 GB RAM.

5.2 Symbolic Proof: Hybrid OPC UA

Having analyzed OPC UA's conventional security handshake, we turn to our first quantum-resistant solution: Hybrid OPC UA. Our ProVerif model of Hybrid OPC UA follows the illustration in Fig. 8. Note that operations and data related to PQC are highlighted in yellow, while operations and data related to the XOR-then-MAC combiner are highlighted in gray.

As our integration of PQC into OPC UA relies on KEM constructions, we have to formalize KEMs according to their definition (see Section 2.2). To the best of our knowledge we are the first to formally describe KEMs in ProVerif. However, a recent analysis of the Hybrid Public Key Encryption scheme provides definitions of authenticated KEMs in CryptoVerif [4]. We adopt their

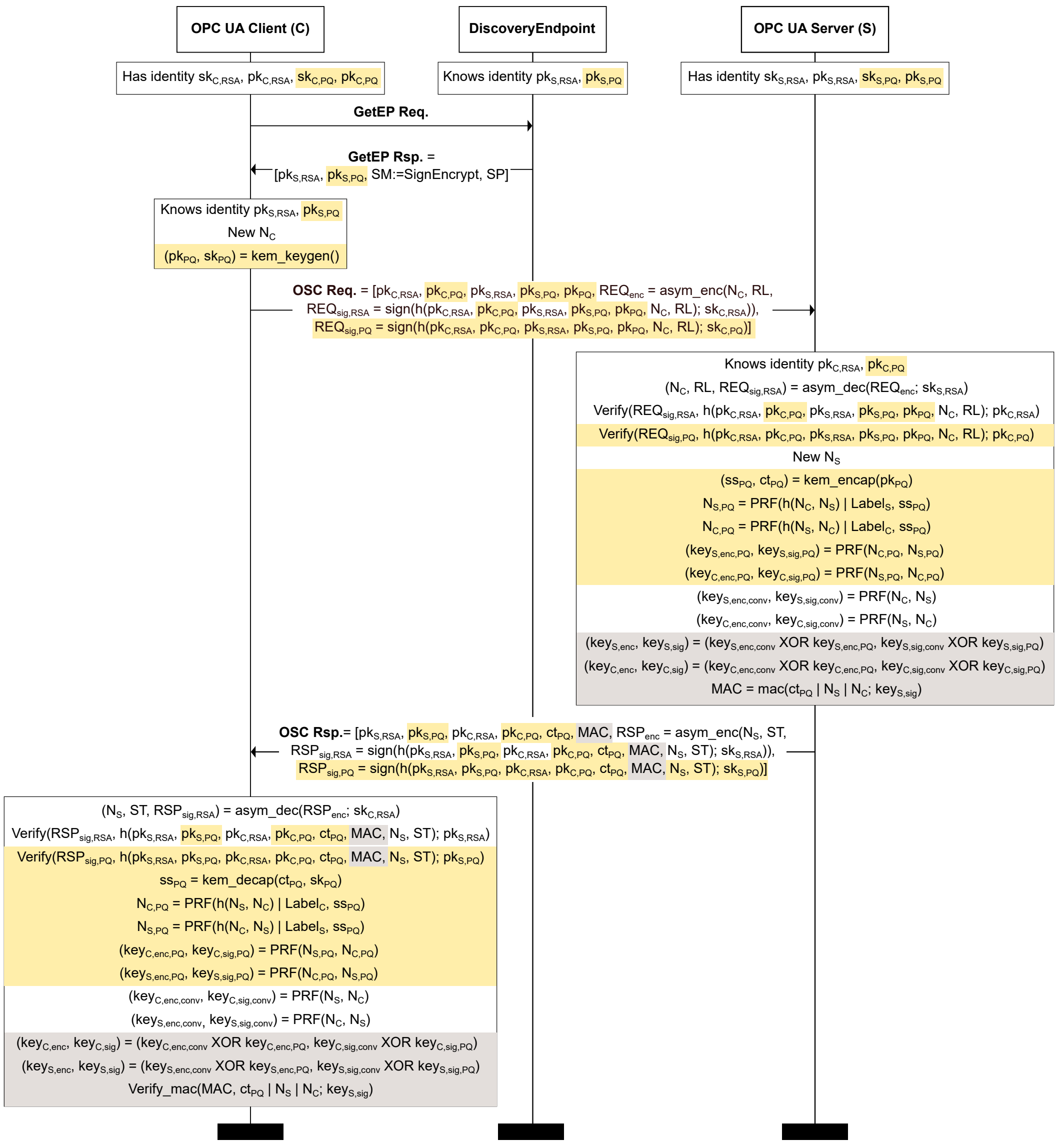

OPC UA Client (C)
DiscoveryEndpoint
OPC UA Server (S)
Has identity $sk_{C,RSA}$, $pk_{C,RSA}$, $sk_{C,PQ}$, $pk_{C,PQ}$
Knows identity $pk_{S,RSA}$, $pk_{S,PQ}$
Has identity $sk_{S,RSA}$, $pk_{S,RSA}$, $sk_{S,PQ}$, $pk_{S,PQ}$
GetEP Req.
GetEP Rsp. = [$pk_{S,RSA}$, $pk_{S,PQ}$, SM:=SignEncrypt, SP]
Knows identity $pk_{S,RSA}$, $pk_{S,PQ}$
New N_C
(pk_{PQ}, sk_{PQ}) = kem_keygen()
OSC Req. = [$pk_{C,RSA}$, $pk_{C,PQ}$, $pk_{S,RSA}$, $pk_{S,PQ}$, pk_{PQ}, REQ_{enc} = asym_enc(N_C, RL, $REQ_{sig,RSA}$ = sign(h($pk_{C,RSA}$, $pk_{C,PQ}$, $pk_{S,RSA}$, $pk_{S,PQ}$, pk_{PQ}, N_C, RL); $sk_{C,RSA}$)), $REQ_{sig,PQ}$ = sign(h($pk_{C,RSA}$, $pk_{C,PQ}$, $pk_{S,RSA}$, $pk_{S,PQ}$, pk_{PQ}, N_C, RL); $sk_{C,PQ}$)]
Knows identity $pk_{C,RSA}$, $pk_{C,PQ}$
(N_C, RL, $REQ_{sig,RSA}$) = asym_dec(REQ_{enc}; $sk_{S,RSA}$)
Verify($REQ_{sig,RSA}$, h($pk_{C,RSA}$, $pk_{C,PQ}$, $pk_{S,RSA}$, $pk_{S,PQ}$, pk_{PQ}, N_C, RL); $pk_{C,RSA}$)
Verify($REQ_{sig,PQ}$, h($pk_{C,RSA}$, $pk_{C,PQ}$, $pk_{S,RSA}$, $pk_{S,PQ}$, pk_{PQ}, N_C, RL); $pk_{C,PQ}$)
New N_S
(ss_{PQ}, ct_{PQ}) = kem_encap(pk_{PQ})
$N_{S,PQ}$ = PRF(h(N_C, N_S) | $Label_S$, ss_{PQ})
$N_{C,PQ}$ = PRF(h(N_S, N_C) | $Label_C$, ss_{PQ})
($key_{S,enc,PQ}$, $key_{S,sig,PQ}$) = PRF($N_{C,PQ}$, $N_{S,PQ}$)
($key_{C,enc,PQ}$, $key_{C,sig,PQ}$) = PRF($N_{S,PQ}$, $N_{C,PQ}$)
($key_{S,enc,conv}$, $key_{S,sig,conv}$) = PRF(N_C, N_S)
($key_{C,enc,conv}$, $key_{C,sig,conv}$) = PRF(N_S, N_C)
($key_{S,enc}$, $key_{S,sig}$) = ($key_{S,enc,conv}$ XOR $key_{S,enc,PQ}$, $key_{S,sig,conv}$ XOR $key_{S,sig,PQ}$)
($key_{C,enc}$, $key_{C,sig}$) = ($key_{C,enc,conv}$ XOR $key_{C,enc,PQ}$, $key_{C,sig,conv}$ XOR $key_{C,sig,PQ}$)
MAC = mac(ct_{PQ} | N_S | N_C; $key_{S,sig}$)
OSC Rsp.= [$pk_{S,RSA}$, $pk_{S,PQ}$, $pk_{C,RSA}$, $pk_{C,PQ}$, ct_{PQ}, MAC, RSP_{enc} = asym_enc(N_S, ST, $RSP_{sig,RSA}$ = sign(h($pk_{S,RSA}$, $pk_{S,PQ}$, $pk_{C,RSA}$, $pk_{C,PQ}$, ct_{PQ}, MAC, N_S, ST); $sk_{S,RSA}$)), $RSP_{sig,PQ}$ = sign(h($pk_{S,RSA}$, $pk_{S,PQ}$, $pk_{C,RSA}$, $pk_{C,PQ}$, ct_{PQ}, MAC, N_S, ST); $sk_{S,PQ}$)]
(N_S, ST, $RSP_{sig,RSA}$) = asym_dec(RSP_{enc}; $sk_{C,RSA}$)
Verify($RSP_{sig,RSA}$, h($pk_{S,RSA}$, $pk_{S,PQ}$, $pk_{C,RSA}$, $pk_{C,PQ}$, ct_{PQ}, MAC, N_S, ST); $pk_{S,RSA}$)
Verify($RSP_{sig,PQ}$, h($pk_{S,RSA}$, $pk_{S,PQ}$, $pk_{C,RSA}$, $pk_{C,PQ}$, ct_{PQ}, MAC, N_S, ST); $pk_{S,PQ}$)
ss_{PQ} = kem_decap(ct_{PQ}, sk_{PQ})
$N_{C,PQ}$ = PRF(h(N_S, N_C) | $Label_C$, ss_{PQ})
$N_{S,PQ}$ = PRF(h(N_C, N_S) | $Label_S$, ss_{PQ})
($key_{C,enc,PQ}$, $key_{C,sig,PQ}$) = PRF($N_{S,PQ}$, $N_{C,PQ}$)
($key_{S,enc,PQ}$, $key_{S,sig,PQ}$) = PRF($N_{C,PQ}$, $N_{S,PQ}$)
($key_{C,enc,conv}$, $key_{C,sig,conv}$) = PRF(N_S, N_C)
($key_{S,enc,conv}$, $key_{S,sig,conv}$) = PRF(N_C, N_S)
($key_{C,enc}$, $key_{C,sig}$) = ($key_{C,enc,conv}$ XOR $key_{C,enc,PQ}$, $key_{C,sig,conv}$ XOR $key_{C,sig,PQ}$)
($key_{S,enc}$, $key_{S,sig}$) = ($key_{S,enc,conv}$ XOR $key_{S,enc,PQ}$, $key_{S,sig,conv}$ XOR $key_{S,sig,PQ}$)
Verify_mac(MAC, ct_{PQ} | N_S | N_C; $key_{S,sig}$)

Fig. 8. Formal model of Hybrid OPC UA in security mode SignAndEncrypt (yellow: operations/data related to PQC; gray: operations/data related to XOR-then-MAC combiner).

model to receive a standard KEM construction in ProVerif. For all other cryptographic primitives, we are able to rely on definitions from ProVerif's official manual [25] and existing, open-source models [16, 68].

In ProVerif, cryptographic primitives are always modeled as deterministic functions. Therefore, we need to transform the probabilistic nature of KEMs into an intrinsically deterministic behavior. For this, we declare two new types: `kem_keypairseed` and `kem_seed`. The former explicitly models the required randomness within key generation `kem_keygen()` and the latter models a random seed within the encapsulation step `kem_encap`, where it serves as additional input to the generation of a

Listing 4. KEM functionality in ProVerif.

```
(* Declarations *)
type kem_keypairseed.
type kem_seed.
type kem_privkey.
type kem_pubkey.
type kem_secret.
type kem_ciphertext.

(* Key Generation() -> (pk, sk) *)
fun kem_pk(kem_keypairseed): kem_pubkey.
fun kem_sk(kem_keypairseed): kem_privkey.

letfun kem_keygen() =
new s:kem_keypairseed;
let keygen_pk = kem_pk(s) in
let keygen_sk = kem_sk(s) in
(keygen_pk, keygen_sk).

(* Encapsulation(pk) -> (ss, ct) *)
fun kem_encap_r_secret(kem_seed, kem_pubkey): kem_secret.
fun kem_encap_r_enc(kem_seed, kem_pubkey): kem_ciphertext.

letfun kem_encap(pk:kem_pubkey) =
new k:kem_seed;
let encap_secret = kem_encap_r_secret(k, pk) in
let encap_ciphertext = kem_encap_r_enc(k, pk) in
(encap_secret, encap_ciphertext).

(* Decapsulation(ct, sk) -> (ss) *)
fun kem_decap(kem_ciphertext, kem_privkey): kem_secret.
equation forall k:kem_seed, ks:kem_keypairseed;
kem_decap(
kem_encap_r_enc(k, kem_pk(ks)),
kem_sk(ks)
) = kem_encap_r_secret(k, kem_pk(ks)).
```

shared secret (`kem_encap_r_secret`) and ciphertext (`kem_encap_r_enc`). As a result, it is ensured that the seeds are freshly chosen at each call to the key generation and encapsulation function. The decapsulation function `kem_decap` is declared as standard destructor that properly captures the relation between decapsulation and encapsulation. Note that because of the symbolic model and the underlying perfect cryptography assumption, the user-defined KEM functions are not capable of modeling potential decryption failures. Treating such failures is not supported in ProVerif at the time of writing. As this has led to certain attack scenarios in some of the proposed post-quantum KEMs, adoptions may be required in the future to address them accordingly. Listing 4 shows the resulting KEM model in ProVerif.

Apart from the integration of a post-quantum KEM we need to model the additional PQC signature scheme (new long-term key pair, additional sign and verify operations), the modified structure of request and response message, the derivation of additional nonces, and the hybrid construction based on the XOR-then-MAC combiner. For the evaluation of our Hybrid OPC UA model, we use the same setup as in the case of conventional OPC UA. ProVerif finishes its analysis within seconds (8.870 s) on our notebook and finds no attacks. Table 1 gives a complete overview of the verification results of the proposed hybrid security handshake.

As our hybrid mode promises confidentiality as long as one of the underlying two primitives remains unbroken (conventional public key cryptography or post-quantum key encapsulation), we also verify this presumption. Accidentally leaking either the keying material derived from classical cryptography or the keying material derived from the quantum-resistant primitives gives

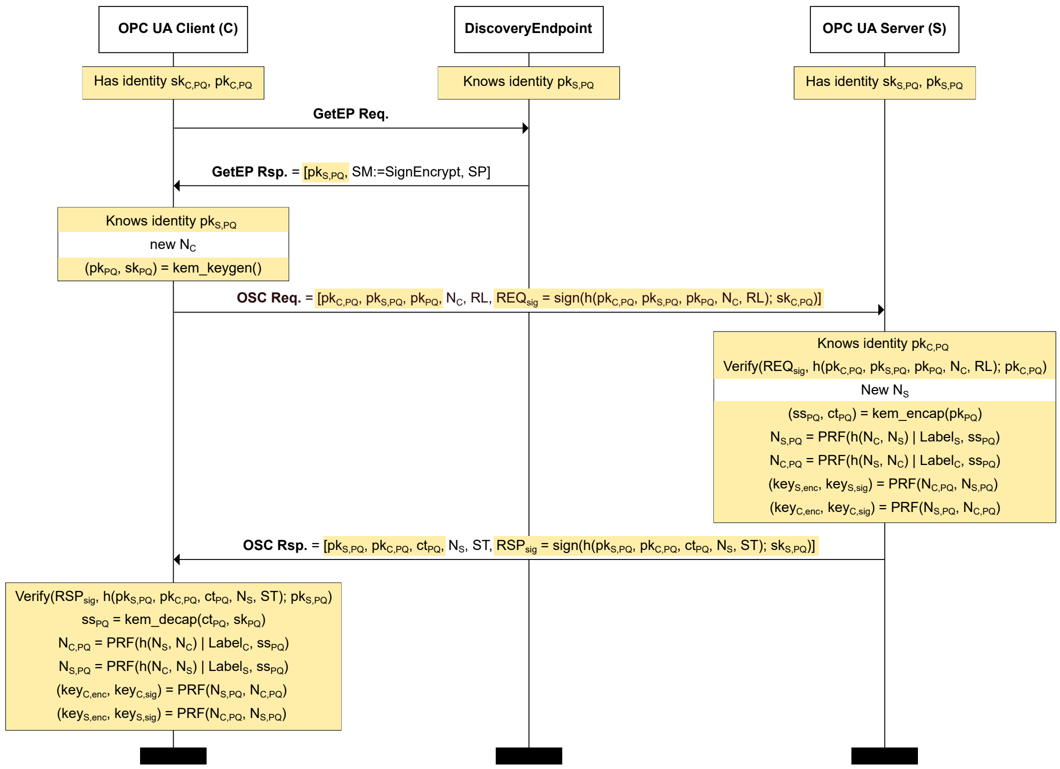

OPC UA Client (C)
DiscoveryEndpoint
OPC UA Server (S)
Has identity $sk_{C,PQ}$, $pk_{C,PQ}$
Knows identity $pk_{S,PQ}$
Has identity $sk_{S,PQ}$, $pk_{S,PQ}$
GetEP Req.
GetEP Rsp. = [$pk_{S,PQ}$, SM:=SignEncrypt, SP]
Knows identity $pk_{S,PQ}$
new N_C
(pk_{PQ}, sk_{PQ}) = kem_keygen()
OSC Req. = [$pk_{C,PQ}$, $pk_{S,PQ}$, pk_{PQ}, N_C, RL, REQ_{sig} = sign(h($pk_{C,PQ}$, $pk_{S,PQ}$, pk_{PQ}, N_C, RL); $sk_{C,PQ}$)]
Knows identity $pk_{C,PQ}$
Verify(REQ_{sig}, h($pk_{C,PQ}$, $pk_{S,PQ}$, pk_{PQ}, N_C, RL); $pk_{C,PQ}$)
New N_S
(ss_{PQ}, ct_{PQ}) = kem_encap(pk_{PQ})
$N_{S,PQ}$ = PRF(h(N_C, N_S) | $Label_S$, ss_{PQ})
$N_{C,PQ}$ = PRF(h(N_S, N_C) | $Label_C$, ss_{PQ})
($key_{S,enc}$, $key_{S,sig}$) = PRF($N_{C,PQ}$, $N_{S,PQ}$)
($key_{C,enc}$, $key_{C,sig}$) = PRF($N_{S,PQ}$, $N_{C,PQ}$)
OSC Rsp. = [$pk_{S,PQ}$, $pk_{C,PQ}$, ct_{PQ}, N_S, ST, RSP_{sig} = sign(h($pk_{S,PQ}$, $pk_{C,PQ}$, ct_{PQ}, N_S, ST); $sk_{S,PQ}$)]
Verify(RSP_{sig}, h($pk_{S,PQ}$, $pk_{C,PQ}$, ct_{PQ}, N_S, ST); $pk_{S,PQ}$)
ss_{PQ} = kem_decap(ct_{PQ}, sk_{PQ})
$N_{C,PQ}$ = PRF(h(N_S, N_C) | $Label_C$, ss_{PQ})
$N_{S,PQ}$ = PRF(h(N_C, N_S) | $Label_S$, ss_{PQ})
($key_{C,enc}$, $key_{C,sig}$) = PRF($N_{S,PQ}$, $N_{C,PQ}$)
($key_{S,enc}$, $key_{S,sig}$) = PRF($N_{C,PQ}$, $N_{S,PQ}$)

Fig. 9. Formal model of PQ OPC UA in security mode SignAndEncrypt (yellow: operations/data related to PQC).

no attacks on the confidentiality properties. Only when both sets of keying material are accessible to the adversary the confidentiality property is broken.

5.3 Symbolic Proof: Post-Quantum OPC UA

For the PQ OPC UA variant, our ProVerif model follows the illustration in Fig. 9; all operations and data related to PQC are highlighted in yellow. As the variant only based on PQC does not require a hybrid construction for key establishment and concatenation of signatures, it is less complex and thus easier to model. In fact, our ProVerif model of PQ OPC UA uses the same KEM definition as in the hybrid case. Having modeled conventional OPC UA and the quantum-resistant primitives for the hybrid approach, it is straightforward to derive the PQ-only variant: Most importantly, the conventional key establishment is replaced by a post-quantum KEM and the conventional signature scheme and its long-term signing key pair is replaced by a corresponding PQC signature scheme. Furthermore, request and response message are now only signed.

Again, the two security properties, i.e. confidentiality and authentication, are successfully verified by ProVerif within seconds (3.086 s) on our notebook. ProVerif's verification results of PQ OPC UA are summarized in Table 1.

5.4 Observation: Symbolic Proofs Based on Verifpal

While ProVerif is considered state of the art and delivers trustworthy results, it is accompanied by a steep learning curve, which makes it hardly accessible to non-expert users. The recently introduced tool Verifpal [46], on the other hand, aims to simplify protocol verification for practitioners [48]. Its modeling language tries to balance between intuitive protocol descriptions and still precise-enough formal models. For instance, in order to avoid user error, it does not allow to model own cryptographic primitives. In addition, it generates analysis output that is easy to interpret. At the time of writing, Verifpal is still considered beta stage software but already produced promising results by verifying complex protocols: the contact tracing protocol DP3T [48] and the secure messaging protocol Signal [46].

As a consequence, we also verified our proposed OPC UA variants using Verifpal to evaluate its accessibility promise. Compared to our initial analysis with ProVerif, Verifpal did not provide any further insights, i.e. find any attacks. While it was straightforward to write the Verifpal models[8] for conventional, hybrid, and post-quantum OPC UA, verification takes significantly longer. For example, verifying the conventional OPC UA model took several days on a powerful remote server, which is equipped with two AMD EPYC 7742 CPUs, each offering 64 cores, running at 2.5 GHz and has 2 TB RAM available.

When comparing the usability of Verifpal to ProVerif, its accessibility to inexperienced users in the domain of protocol analysis is impressive. But, we expect that it will be hard for Verifpal to achieve similar verification speeds as ProVerif while maintaining its low entry barriers (not mentioning the 15+ years of development ProVerif has in advance). Nevertheless, addressing this performance gap seems to be an important target for the first full release.

6 EXPERIMENTAL RESULTS AND EVALUATION

6.1 Implementation Notes

We rely on an open-source OPC UA stack, open62541 [61], to implement our two solutions. Integration of hybrid key exchange, hybrid authentication, and hybrid signatures requires significant changes to the codebase of open62541. To allow for backward compatibility with non-hybrid aware nodes we implement a new security policy *Hybrid{1,3,5}_Basic256*. We add the respective parts of the hybrid key exchange based on KEMs to the client and server code. The key derivation function is adapted to generate two sets of keying material and to combine these two sets using XOR. For our KEM combiner construction, the MAC creation and verification is added as part of the hybrid key exchange. The handling of hybrid authentication based on certificates is integrated and hybrid signature creation and verification is added to the source code. The quantum-resistant signature is appended to the message buffer (not encrypted), while the additional PQ public key and ciphertext of the respective KEM and MAC-value are added to the security header. Our PQ solution requires fewer modifications and uses the new security policy *PQ{1,3,5}*. The KEM-based key exchange is integrated in client and server code. In addition, the generation and verification of PQ signatures and the verification of PQ certificates is implemented. The handling of request and response message needs to be adapted accordingly.

Available tools for generating hybrid certificates either make use of combiners that are not fully backward compatible [73] or implement only a small subset of PQ schemes [18]. Because of these limitations, we implement a new software package capable of creating hybrid and PQ certificates. Our software is capable of creating the X.509 certificate structure from scratch and can freely modify the desired fields. In our case, we rely on two non-critical extensions for storing the additional public key and signature. open62541 uses the cryptographic library mbedTLS for all security relevant functions including the verification of certificates. Therefore, the certificate chain and the trusted root certificates are passed to the verification function provided by mbedTLS.

[8]All implemented models are available at https://github.com/boschresearch/pq_opc-ua_formal_analysis/tree/master/02_verifpal-models; the used version of Verifpal is 0.21.4.

Table 2. Message and certificate sizes for both solutions (in bytes).

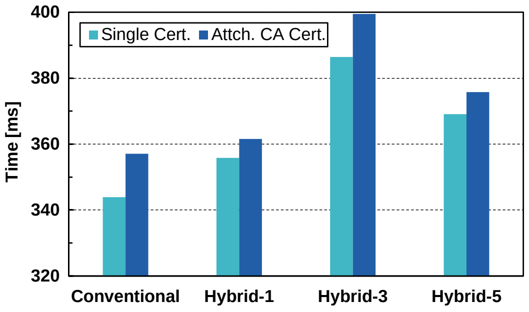
	Solution	*OSC Request* Single Cert.	*OSC Request* Attached CA Cert.	*OSC Response* Single Cert.	*OSC Response* Attached CA Cert.	*Certificate Chain* Single Cert.	*Certificate Chain* Attached CA Cert.
	Conventional (RSA2048)	1,597	2,373	1,601	2,377	908	1,750
Hybrid	1 (Kyber512 + Falcon512 + RSA2048)	4,698	7,147	4,670	7,119	2,515	4,964
Hybrid	3 (Kyber768 + Dilithium4 + RSA2048)	11,945	17,929	11,885	17,869	6,050	12,034
Hybrid	5 (Kyber1024 + Falcon1024 + RSA2048)	7,770	11,755	7,806	11,791	4,051	8,036
PQ	1 (Kyber512 + Falcon512)	3,618	5,472	3,593	5,447	1,924	3,778
PQ	3 (Kyber768 + Dilithium4)	10,211	15,598	10,154	15,541	5,457	10,844
PQ	5 (Kyber1024 + Falcon1024)	6,562	9,952	6,601	9,991	3,460	6,850

We are able to use this function without modifications since our generated hybrid certificates are fully compliant to the X.509 standard. The verification function of mbedTLS allows to provide an optional callback function as parameter that is called after each certificate in the chain was verified. We use this callback mechanism to verify the additional PQ signature inside the custom extension of our hybrid certificates. It should be noted that verification of PQ certificates takes place outside mbedTLS since we did not integrate our selected PQ schemes into the embedded TLS library. Instead, we rely on its mechanism to parse encoded certificates, which required minor changes to mbedTLS because of unique algorithm identifiers used in our PQ X.509 certificates.

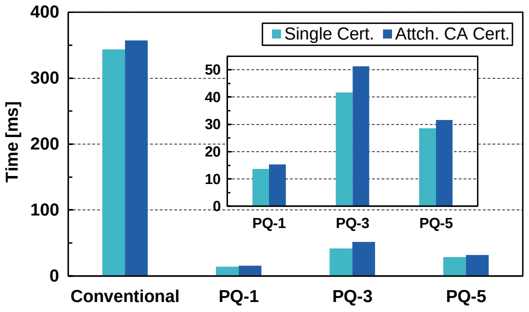
6.2 Measurement Setup

Our setup resembles a typical use case for OPC UA within an industrial network: Two CPS (e.g. control unit and gateway) wish to securely exchange data, which requires the establishment of a secure channel. We select the Raspberry Pi 3 Model B as our evaluation platform. It features a 1.2 GHz quad-core CPU (ARM Cortex-A53), 1024 MB RAM, and requires a SD-card to store operating system and software. As affordable single-board computer Raspberry Pis have become very popular prototyping platforms even for industrial use cases [69]. The two Raspberry Pis are connected to the same network via their 100 Mbit Ethernet interfaces, one is instantiated as OPC UA client and the other as OPC UA server. For our timing measurements, we rely on the same kernel extensions introduced in *Preliminary Performance Evaluation* (see Section 4.3.2). Since our measurements also include network round-trip time and overhead of the network stack, we report the time elapsed until completion of the OPC UA handshake in milliseconds. Therefore, we convert the cycle counts obtained from the two Raspberry Pis to milliseconds.

Besides complete handshake duration, we report the performance of OPC UA's security handshake in terms of message and certificate size. Our baseline measurement considers a conventional OPC UA security handshake using security policy *Basic256Sha256*. Both solutions are evaluated at three NIST security levels (see Section 4.3). This leads to a total of six different test cases: *Hybrid-{1,3,5}* and *PQ-{1,3,5}*. In addition, we evaluate each test case in two different scenarios regarding included certificates. In the first scenario, only a single device certificate (*Single Cert.*) is conveyed. The second scenario assumes that OPC UA client and server are part of a larger industrial network containing intermediate certificate authorities (CA). In this case, the certificate chain contains the device and one attached intermediate CA certificate (*Attch. CA Cert.*). For each of the above test cases and the two scenarios, we record the establishment of 100 secure channels and report the average values.



(a) Hybrid OPC UA.



(b) Post-Quantum OPC UA.

Fig. 10. Comparison of average handshake duration at different security levels.

6.3 Results and Evaluation

6.3.1 Hybrid OPC UA. Table 2 shows the impact of our hybrid security handshake on the size of the *OSC Req.* and *OSC Rsp.* message at different security levels. Besides that, certificate sizes for both scenarios are reported. As expected, because of the hybrid mode, the message sizes increase at all levels. The highest increment compared to conventional OPC UA can be observed at security level 3: In case an additional CA certificate is attached, the size of the *OSC Req.* and *OSC Rsp.* message increases in average 7.5 times. Considering certificate sizes, the smallest increase is observed with certificates containing an additional Falcon512 public key and signature (factor of 2.8).

Figure 10(a) shows the results of the conducted performance measurements. As expected, the duration of the handshake increases at all security levels. However, the most time during the handshake is spent conventionally decrypting and signing the request and response message. In case a single hybrid certificate is conveyed, the fastest observed hybrid handshake adds only 11.9 ms to the total duration (Hybrid-1), while the slowest leads to an overhead of 42.6 ms (Hybrid-3). The extra time spent verifying an attached intermediate CA certificate is clearly visible in Fig. 10(a) and correlates to the reported verification times in Fig. 6. Since our implementation of Falcon makes use of floating-point operations, the overhead in Hybrid-1 and Hybrid-5 remains very small. Because both nodes are connected via fast network interfaces, the larger message sizes have only little impact on the total duration of the handshake: Sending the response and request message in Hybrid-3 with an intermediate CA certificate attached takes 0.4 ms.

6.3.2 Post-Quantum OPC UA. Table 2 also shows the message and certificate sizes for our solution solely based on PQC. Similar to our hybrid solution, we observe that all message sizes as well as certificate sizes increase at all security levels due to the larger public keys and signatures of the integrated PQ schemes. Besides that, instantiations using Falcon show a significantly lower overhead.

The results of our performance measurements (see Fig. 10(b)), however, show a significant improvement compared to OPC UA's conventional security handshake. Across all security levels our PQ solution is in average 11.5 times faster than conventional OPC UA. The fact that we omit all cryptographic operations based on RSA from OPC UA's conventional security handshake substantially increases its performance. With a handshake duration of just 28.6 ms, PQ-5 (single certificate) is even faster than PQ-3 (41.8 ms). As the signature generation and verification times of Falcon and Dilithium are generally slower than Kyber's KEM functions, client and server spend most of the time during the handshake performing operations of the respective DSA. For example, deriving symmetric keying material requires 3.5 ms compared to 10.2 ms spent on the creation and verification of signatures in PQ-1. Similar to our hybrid approach, message sizes have only little impact on the overall handshake duration.

Both our solutions demonstrate that Falcon is preferable over Dilithium in case both communicating nodes are capable of using its efficient floating-point arithmetic. Our Hybrid-5 and PQ-5 solutions even lead to significantly less overhead – in terms of handshake duration and size – than Hybrid-3 and PQ-3. Since message sizes do not negatively impact the performance of the security handshake as much as slower algorithms do, we recommend to use Dilithium2 in case security level 1 is required and floating-point support cannot be assumed.

7 CONCLUSION

In this work, we proposed two novel solutions for the integration of post-quantum primitives, i.e. key establishment and digital signatures, into the security handshake of the industrial M2M protocol OPC UA. Our first solution considers hybrid key exchange, hybrid authentication, and hybrid signatures, while the second is solely based on quantum-resistant primitives. Compared to other previous works, this approach allows to investigate the total impact of post-quantum cryptography. Furthermore, we formally verified confidentiality and authentication properties of the proposed integrations in the symbolic model. These symbolic proofs are realized using the state-of-the-art verification tool ProVerif.

Alongside the description and formal analysis of our two solutions, we selected three algorithms based on an investigation of all lattice-based schemes submitted to NIST's PQC standardization process. Subsequently, we instantiated our two solutions at three different NIST security levels using the respective parameter sets of Kyber{512,768,1024} for key establishment and Falcon{512,1024}-FPU or Dilithium4 for digital signatures. In our performance measurements, we compared the handshake duration of both solutions to that of conventional OPC UA for different security levels and certificate scenarios. Our hybrid integration leads to acceptable overhead in terms of latency and message sizes, while our PQ solution significantly outperforms conventional OPC UA at all security levels in terms of handshake duration. OPC UA provides mutual authentication based on X.509 certificates. Our hybrid solution works with hybrid certificates using non-critical extension fields to achieve backward compatibility with non-hybrid aware clients and servers. Furthermore, our described verification of hybrid certificates using mbedTLS applies to use cases outside the industrial domain. Ultimately, our two solutions provide comprehensive insights into the feasibility of integrating PQC into OPC UA and demonstrate that PQC is practical for ICS. Falcon and Dilithium are efficient options for PQ signature schemes; in case floating-point support is available, Falcon provides faster performance at smaller public key and signature sizes. In our two solutions, Kyber showed very efficient performance throughout all evaluated security levels.

As future work, we will continue to investigate our two solutions, particularly with regard to potential optimizations for time-sensitive industrial applications. In addition, we plan to evaluate our proposed solutions in industrial networks under more realistic conditions. In order to provide additional security guarantees, we will analyze our proposed variants in the computational model with less idealized assumptions using a semi-automatic prover such as CryptoVerif.

ACKNOWLEDGMENTS

The authors would like to thank Benjamin Lipp and Nadim Kobeissi for their helpful discussions and support in detecting and fixing a bug in the Verifpal model. Furthermore, we appreciate the allocated runtime for our Verifpal verifications on the computing server of the chair of cryptography and IT-security, Ruhr University Bochum.

The work presented in this paper has been partly funded by the German Federal Ministry of Education and Research (BMBF) under the project “FLOQI” (ID 16KIS1074).

REFERENCES

[1] Martín Abadi, Bruno Blanchet, and Cédric Fournet. 2017. The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication. *J. ACM* 65 (2017), 41 pages. Issue 1. https://doi.org/10.1145/3127586

[2] Erdem Alkim, Roberto Avanzi, Joppe W. Bos, Léo Ducas, Antonio De La Piedra, Thomas Pöppelmann, Peter Schwabe, Douglas Stebila, Martin R. Albrecht, Emmanuela Orsini, Valery Osheter, Kenneth G. Paterson, Guy Peer, and Nigel P. Smart. 2019. NewHope. NIST PQC Standardization: Round 2.

[3] Erdem Alkim, Joppe W. Bos, Léo Ducas, Patrick Longa, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Chris Peikert, Ananth Raghunathan, and Douglas Stebila. 2020. FrodoKEM: Learning With Errors Key Encapsulation. NIST PQC Standardization: Round 2.

[4] Joël Alwen, Bruno Blanchet, Eduard Hauck, Eike Kiltz, Benjamin Lipp, and Doreen Riepel. 2021. Analysing the HPKE Standard. In *Advances in Cryptology – EUROCRYPT 2021*, Anne Canteaut and François-Xavier Standaert (Eds.). Springer, 87–116. https://doi.org/10.1007/978-3-030-77870-5_4

[5] Matthew Arcus. 2018. *Using the Cycle Counter Registers on the Raspberry Pi 3.* https://matthewarcus.wordpress.com/2018/01/27/using-the-cycle-counter-registers-on-the-raspberry-pi-3

[6] Arm Limited. 2020. Arm Architecture Reference Manual: Armv8. https://developer.arm.com/documentation/ddi0487/latest/

[7] Frank Arute, Kunal Arya, Ryan Babbush, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574 (2019), 505–510. https://doi.org/10.1038/s41586-019-1666-5

[8] Roberto Avanzi, Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schank, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2019. CRYSTALS-Kyber. NIST PQC Standardization: Round 2.

[9] AWS Blog. 2019. *Converting industrial protocols with AWS IoT Greengrass.* https://aws.amazon.com/de/blogs/iot/converting-industrial-protocols-with-aws-iot-greengrass/

[10] Hayo Baan, Sauvik Bhattacharya, Scott Fluhrer, Oscar Garcia-Morchon, Thijs Laarhoven, Rachel Player, Ronald Rietmann, Markku-Juhani O. Saarinen, Ludo Tolhuizen, José Luis Torre-Arce, and Zhenfei Zhang. 2019. Round5. KEM and PKE based on (Ring) Learning with Rounding. NIST PQC Standardization: Round 2.

[11] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. 2021. SoK: Computer-Aided Cryptography. In *2021 IEEE Symposium on Security and Privacy (SP).* IEEE, 777–795. https://doi.org/10.1109/SP40001.2021.00008

[12] Elaine Barker, Lily Chen, and Richard Davis. 2020. *Recommendation for Key-Derivation Methods in Key-Establishment Schemes.* NIST. https://doi.org/10.6028/NIST.SP.800-56Cr2

[13] David Basin, Cas Cremers, Jannik Dreier, Simon Meier, Ralf Sasse, and Benedikt Schmidt. 2021. Tamarin-Prover Manual. https://tamarin-prover.github.io/manual/book/001_introduction.html

[14] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirovic, Ralf Sasse, and Vincent Stettler. 2018. A Formal Analysis of 5G Authentication. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18).* ACM, 1383–1396. https://doi.org/10.1145/3243734.3243846

[15] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine Van Vredendaal. 2019. NTRU Prime. Round 2. NIST PQC Standardization: Round 2.

[16] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. 2017. Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate. In *2017 IEEE Symposium on Security and Privacy (SP).* IEEE, 483–502. https://doi.org/10.1109/SP.2017.26

[17] Nina Bindel, Sedat Akleylek, Erdem Alkim, Paulo S. L. M. Bareto, Johannes Buchmann, Edward Eaton, Gutoski Gus, Juliane Krämer, Patrick Longa, Harun Polat, Jefferson E. Ricardini, and Gustavo Zanon. 2019. qTESLA. NIST PQC Standardization: Round 2.

[18] Nina Bindel, Johannes Braun, Luca Gladiator, Tobias Stöckert, and Johannes Wirth. 2019. X.509-Compliant Hybrid Certificates for the Post-Quantum Transition. *Journal of Open Source Software* 4 (2019). Issue 40. https://doi.org/10.21105/joss.01606

[19] Nina Bindel, Jacqueline Brendel, Marc Fischlin, Brian Concalves, and Douglas Stebila. 2019. Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange. In *Post-Quantum Cryptography. PQCrypto 2019* (Cham) *(Lecture notes in computer science, 11505)*, Jintai Ding and Rainer Steinwandt (Eds.). Springer International Publishing, 206–226. https://doi.org/10.1007/978-3-030-25510-7_12

[20] Nina Bindel, Udyani Herath, Matthew McKague, and Douglas Stebila. 2017. Transitioning to a Quantum-Resistant Public Key Infrastructure. In *Post-Quantum Cryptography. PQCrypto 2017* (Cham) *(Lecture notes in computer science, 10346)*, Tanja Lange and Tsuyoshi Takagi (Eds.). Springer International Publishing, 384–405. https://doi.org/10.1007/978-3-319-59879-6_22

[21] Bruno Blanchet. 2001. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14).* IEEE, 82–96. https://doi.org/10.1109/CSFW.2001.930138

[22] Bruno Blanchet. 2012. Security Protocol Verification: Symbolic and Computational Models. In *Principles of Security and Trust (POST 2012)*, Pierpaolo Degano and Joshua D. Guttman (Eds.). Springer, 3–29. https://doi.org/10.1007/978-3-642-28641-4_2

[23] Bruno Blanchet. 2016. Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. *Foundations and Trends in Privacy and Security* 1 (2016), 1–135. Issue 1–2. https://doi.org/10.1561/3300000004

[24] Bruno Blanchet. 2017. CryptoVerif. A Computationally-Sound Security Protocol Verifier.

[25] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. [n. d.]. ProVerif 2.02pl1: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial.

[26] Matt Braithwaite. 2016. *Experimenting with Post-Quantum Cryptography.* Google. https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html

[27] BSI. 2017. OPC UA Security Analysis.

[28] BSI. 2020. Migration zu Post-Quanten-Kryptografie. [Migration to Post-Quantum Cryptography]. (available only in German).

[29] Kevin Bürstinghaus-Steinbach, Christoph Krauß, Ruben Niederhagen, and Michael Schneider. 2020. Post-Quantum TLS on Embedded Systems. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (ASIA CCS '20)*. ACM, 841–852. https://doi.org/10.1145/3320269.3384725

[30] Matt Campagna and Eric Crockett. 2019. *Hybrid Post-Quantum Key Encapsulation Methods (PQ KEM) for Transport Layer Security 1.2 (TLS).* Internet-Draft draft-campagna-tls-bike-sike-hybrid-01. IETF. https://datatracker.ietf.org/doc/html/draft-campagna-tls-bike-sike-hybrid-01 (Work in progress).

[31] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hülsing, Joost Rijneveld, John M. Schank, Peter Schwabe, William Whyte, and Zhenfei Zhang. 2019. NTRU. NIST PQC Standardization: Round 2.

[32] Vincent Cheval and Bruno Blanchet. 2013. Proving More Observational Equivalences with ProVerif. In *Principles of Security and Trust (POST 2013)*, David Basin and John C. Mitchell (Eds.). Springer, 226–246. https://doi.org/10.1007/978-3-642-36830-1_12

[33] Eric Crockett, Christian Paquin, and Douglas Stebila. 2019. Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH. *Cryptology ePrint Archive, Report 2019/858* (2019). https://ia.cr/2019/858

[34] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. 2019. SABER: Mod-LWR based KEM. NIST PQC Standardization: Round 2.

[35] Jan-Pieter D'Anvers, Marcel Tiepelt, Frederik Vercauteren, and Ingrid Verbauwhede. 2019. Timing Attacks on Error Correcting Codes in Post-Quantum Schemes. In *Proceedings of ACM Workshop on Theory of Implementation Security Workshop (TIS '19)*. ACM, 2–9. https://doi.org/10.1145/3338467.3358948

[36] D. Dolev and A. Yao. 1983. On the security of public key protocols. *IEEE Transactions on Information Theory* 29 (1983), 198–208. Issue 2. https://doi.org/10.1109/TIT.1983.1056650

[37] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2019. CRYSTALS-Dilithium. Algorithm Specifications and Supporting Documentation. NIST PQC Standardization: Round 2.

[38] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, Wiliam Whyte, and Zhenfei Zhang. 2019. Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU. NIST PQC Standardization: Round 2.

[39] Qian Guo, Thomas Johansson, and Jing Yang. 2019. A Novel CCA Attack Using Decryption Errors Against LAC. In *Advances in Cryptology – ASIACRYPT 2019*, Steven D. Galbraith and Shiho Moriai (Eds.). Springer, 82–111. https://doi.org/10.1007/978-3-030-34578-5_4

[40] Mike Hamburg. 2019. ThreeBears. Post-quantum cryptography proposal. NIST PQC Standardization: Round 2.

[41] Andreas Hülsing, Kai-Chun Ning, Peter Schwabe, Florian Weber, and Philip R. Zimmermann. 2021. Post-quantum WireGuard. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 304–321. https://doi.org/10.1109/SP40001.2021.00030

[42] Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. 2015. On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 v1.5 Encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. ACM, 1185–1196. https://doi.org/10.1145/2810103.2813657

[43] Panos Kampanakis, Peter Panburana, Ellie Daw, and Daniel Van Geest. 2018. The Viability of Post-Quantum X.509 Certificates. *Cryptology ePrint Archive, Report 2018/063* (2018). https://ia.cr/2018/063

[44] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. 2019. pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4. *Cryptology ePrint Archive, Report 2019/844* (2019). https://ia.cr/2019/844

[45] Nadim Kobeissi. 2018. *Formal Verification for Real-World Cryptographic Protocols and Implementations.* PhD thesis. Ecole Normale Supérieure de Paris.

[46] Nadim Kobeissi. 2020. Verifpal: User Manual.

[47] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. 2017. Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach. In *2017 IEEE European Symposium on*

Security and Privacy (EuroSP). IEEE, 435–450. https://doi.org/10.1109/EuroSP.2017.38

[48] Nadim Kobeissi, Georgio Nicolas, and Mukesh Tiwari. 2020. Verifpal: Cryptographic Protocol Analysis for the Real World. In *Progress in Cryptology – INDOCRYPT 2020*, Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran (Eds.). Springer, 151–202. https://doi.org/10.1007/978-3-030-65277-7_8

[49] Kris Kwiatkowski and Luke Valenta. 2019. *The TLS Post-Quantum Experiment*. Cloudflare. https://blog.cloudflare.com/the-tls-post-quantum-experiment/

[50] Timm Lauser, Daniel Zelle, and Christoph Krauß. 2020. Security Analysis of Automotive Protocols. In *Computer Science in Cars Symposium (CSCS '20)*. ACM. https://doi.org/10.1145/3385958.3430482

[51] Gavin Lowe. 1996. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1996)*, Tiziana Margaria-Steffen and Bernhard Steffen (Eds.). Springer, 147–166. https://doi.org/10.1007/3-540-61042-1_43

[52] David McGrew. 2017. Cryptographic Agility in the Real World. In *Cryptographic Agility and Interoperability: Proceedings of a Workshop*, National Academies of Sciences, Engineering, and Medicine (Ed.). The National Academies Press, 34–38.

[53] Daniele Micciancio and Oded Regev. 2008. Lattice-based Cryptography. In *Post-Quantum Cryptography*, Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen (Eds.). Springer, 146–191. https://doi.org/10.1007/978-3-540-88702-7_5

[54] Microsoft Azure. 2021. *What is Industrial IoT (IIoT)?* https://docs.microsoft.com/en-us/azure/industrial-iot/overview-what-is-industrial-iot

[55] J. C. Mitchell, M. Mitchell, and U. Stern. 1997. Automated analysis of cryptographic protocols using Murφ. In *1997 IEEE Symposium on Security & Privacy (SP)*. IEEE, 141–151. https://doi.org/10.1109/SECPRI.1997.601329

[56] Roger M. Needham and Michael D. Schroeder. 1978. Using Encryption for Authentication in Large Networks of Computers. *Commun. ACM* 21 (1978), 993–999. Issue 12. https://doi.org/10.1145/359657.359659

[57] OPC Foundation. 2017. OPC UA Specification. Part 1 - Overview and Concepts Release 1.04.

[58] OPC Foundation. 2017. OPC UA Specification. Part 4 - Services Release 1.04.

[59] OPC Foundation. 2017. OPC UA Specification. Part 6 - Mappings Release 1.04.

[60] OPC Foundation. 2020. OPC UA Roadmap. https://opcfoundation.org/about/opc-technologies/opc-ua/opcua-roadmap/

[61] Florian Palm, Sten Grüner, Julius Pfrommer, Markus Graube, and Leon Urbas. 2015. Open source as enabler for OPC UA in industrial automation. In *2015 20th IEEE Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE, 1–6. https://doi.org/10.1109/ETFA.2015.7301562

[62] Christian Paquin, Douglas Stebila, and Goutam Tamvada. 2020. Benchmarking Post-quantum Cryptography in TLS. In *Post-Quantum Cryptography (PQCrypto 2020)*, Jintai Ding and Jean-Pierre Tillich (Eds.). Springer, 72–91. https://doi.org/10.1007/978-3-030-44223-1_5

[63] Sebastian Paul and Melanie Niethammer. 2019. On the Importance of Cryptographic Agility for Industrial Automation. *at - Automatisierungstechnik (Special Issue: IT-Security in Automation Technology)* 67 (2019), 402–416. Issue 5. https://doi.org/10.1515/auto-2019-0019

[64] Sebastian Paul and Patrik Scheible. 2020. Towards Post-Quantum Security for Cyber-Physical Systems: Integrating PQC into Industrial M2M Communication. In *Computer Security – ESORICS 2020*, Liqun Chen, , Ninghui Li, Kaitai Liang, and Steve Schneider (Eds.). Springer, 295–316. https://doi.org/10.1007/978-3-030-59013-0_15

[65] Thomas Pornin. 2019. PQClean - Falcon implementations (integer-only code, constant-time). https://github.com/PQClean/PQClean/pull/210#issuecomment-513827611

[66] Stefan Profanter, Ayhun Tekat, Kirill Dorofeev, Markus Rickert, and Alois Knoll. 2019. OPC UA versus ROS, DDS, and MQTT. In *2019 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 955–962. https://doi.org/10.1109/ICIT.2019.8755050

[67] Maxime Puys. 2018. Cybersecurity of Industrial Systems: Applicative Filtering and Generation of Attack Scenarios. PhD thesis defense.

[68] Maxime Puys, Marie-Laure Potet, and Pascal Lafourcade. 2016. Formal Analysis of Security Properties on the OPC-UA SCADA Protocol. In *Computer Safety, Reliability, and Security (SAFECOMP 2016)*, Amund Skavhaug, Jérémie Guiochet, and Friedemann Bitsch (Eds.). Springer, 67–75. https://doi.org/10.1007/978-3-319-45477-1_6

[69] Sfera Labs. 2020. Strato Pi: Industrial Raspberry Pi. https://www.sferalabs.cc/strato-pi/

[70] Peter W. Shor. 1997. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* 26 (1997), 1484–1509. Issue 5. https://doi.org/10.1137/S0097539795293172

[71] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. 2020. Post-Quantum Authentication in TLS 1.3. In *Network and Distributed Systems Security (NDSS) Symposium 2020*. https://doi.org/10.14722/ndss.2020.24203

[72] Douglas Stebila, Scott Fluhrer, and Shay Gueron. 2019. *Hybrid key exchange in TLS 1.3*. Internet-Draft draft-ietf-tls-hybrid-design-01. IETF. https://datatracker.ietf.org/doc/html/draft-ietf-tls-hybrid-design-01 (Work in progress).

[73] Douglas Stebila and Michele Mosca. 2017. Post-quantum Key Exchange for the Internet and the Open Quantum Safe Project. In *Selected Areas in Cryptography – SAC 2016*, Roberto Avanzi and Howard Heys (Eds.). Springer, 14–37. https://doi.org/10.1007/978-3-319-69453-5_2

[74] Mathy Vanhoef and Frank Piessens. 2017. Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, 1313–1328. https://doi.org/10.1145/3133956.3134027

[75] Verizon. 2020. DBIR - Data Breach Investigations Report.

[76] Christoph Weidenbach. 1999. Towards an Automatic Analysis of Security Protocols in First-Order Logic. In *Automated Deduction – CADE-16*, H. Ganzinger (Ed.). Springer, 314–328. https://doi.org/10.1007/3-540-48660-7_29

[77] Martin Wollschlaeger, Thilo Sauter, and Juergen Jasperneite. 2017. The Future of Industrial Communication. *IEEE Industrial Electronics Magazine* 11 (2017), 17–27. Issue 1. https://doi.org/10.1109/MIE.2017.2649104

[78] Lu Xianhui, Liu Yamin, Jia Dingding, Xue Haiyang, He Jingnan, Zhang Zhenfei, Liu Zhe, Yang Hao, Li Bao, and Wang Kunpeng. 2019. LAC. Lattice-based Cryptosystems. NIST PQC Standardization: Round 2.

[79] Jingjing Zhang, Lin Yang, Xianming Gao, and Qiang Wang. 2020. Formal analysis of QUIC handshake protocol using ProVerif. In *2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*. IEEE, 132–138. https://doi.org/10.1109/CSCloud-EdgeCom49738.2020.00030

ALGORITHM OVERVIEW

Table 3. Conventional and PQC signature schemes evaluated in this work.

DSA	*NIST Category*	*Intractable Problem*	*Classical Security*	*PQ Security*	*sk (byte)*	*pk (byte)*	*signature (byte)*
RSA2048	< 1	Integer Factorization	112 bit	–	256	259	256
Dilithium2	1	Module LWE	100 bit	91 bit	2800	1184	2044
Falcon512		NTRU	114 bit	103 bit	1281	897	690
qTESLAp-I		Ring LWE	151 bit	140 bit	5184	14880	2592
SECP256R1		EC Discrete Logarithm	128 bit	–	32	65	73
Dilithium3	2	Module LWE	141 bit	128 bit	3504	1472	2701
Dilithium4	3	ModuleLWE	174 bit	158 bit	3856	1760	3366
qTESLAp-III		Ring LWE	305 bit	279 bit	12352	38432	5664
Falcon1024	5	NTRU	263 bit	230 bit	2305	1793	1330

Table 4. Conventional and PQC key establishment schemes evaluated in this work.

KEM	*NIST Category*	*Intractable Problem*	*Classical Security*	*PQ Security*	*sk (bytes)*	*pk (bytes)*	*ct (bytes)*	*Failure Rate*
Frodo640	1	LWE	144 bit	103 bit	19888	9616	9720	2^{-139}
Kyber512		Module LWE	111 bit	100 bit	1632	800	736	2^{-178}
LAC-128		Ring LWE	147 bit	133 bit	512	544	712	2^{-116}
LightSaber		Module LWR	125 bit	114 bit	1568	672	736	2^{-120}
NewHope512		Ring LWE	112 bit	101 bit	1888	928	1120	2^{-213}
NTRU-HRSS		NTRU	136 bit	123 bit	1450	1138	1138	–
R5ND-1CCA-0d		General LWR	125 bit	115 bit	16	676	740	2^{-157}
SECP256R1		EC Discrete Logarithm	128 bit	–	32	65	65	–
SNTRUP653		NTRU	129 bit	117 bit	1518	994	897	–
BabyBear	2	ModuleLWE	154 bit	140 bit	40	804	917	2^{-156}
Frodo976	3	LWE	209 bit	150 bit	31296	15632	15744	2^{-200}
Kyber768		Module LWE	181 bit	164 bit	2400	1184	1088	2^{-164}
LAC-192		Ring LWE	286 bit	259 bit	1024	1056	1188	2^{-143}
R5ND-3CCA-0d		General LWR	186 bit	174 bit	16	983	1103	2^{-154}
Saber		Module LWR	203 bit	185 bit	2304	992	1088	2^{-136}
SNTRUP761		NTRU	153 bit	139 bit	1763	1158	1039	–
MamaBear	4	Module LWE	235 bit	213 bit	40	1194	1307	2^{-206}
FireSaber	5	Module LWR	283 bit	257 bit	3040	1312	1472	2^{-165}
Frodo1344		LWE	274 bit	196 bit	43088	21520	21632	2^{-253}
Kyber1024		Module LWE	254 bit	230 bit	3168	1568	1568	2^{-174}
LAC-256		Ring LWE	320 bit	290 bit	1024	1056	1424	2^{-122}
NewHope1024		Ring LWE	257 bit	233 bit	3680	1824	2208	2^{-216}
PapaBear		Module LWE	314 bit	280 bit	40	1584	1697	2^{-256}
R5ND-5CCA-0d		General LWR	253 bit	238 bit	16	1349	1509	2^{-145}
SNTRUP857		NTRU	175 bit	159 bit	1999	1322	1184	–