Post-Quantum Authentication in OpenSSL with Hash-Based Signatures

Denis Butin, Julian Wälde, and Johannes Buchmann
TU Darmstadt, Germany
Email: {dbutin, jwaelde, buchmann}@cdc.informatik.tu-darmstadt.de

***Abstract*—Quantum computing is a major threat to contemporary security mechanisms. As standards bodies increasingly focus on post-quantum cryptography, hash-based signatures in particular are often mentioned as a viable solution for quantum-safe authentication. Uniquely, such schemes only require minimal security assumptions. While their security has been analysed thoroughly, their concrete integration in popular security protocols has not been addressed so far. In this paper, we describe our integration of the XMSS hash-based signature scheme into the popular OpenSSL security library. In particular, we introduce support for EVP, ASN.1 and X.509 formats in OpenSSL and for the widely-deployed TLS and S/MIME protocols. Since OpenSSL is sparsely documented, our account can be used as a guide to integrating new signature schemes into the library. Beyond this core integration, we analyse real-world constraints for these protocols, taking into account scheme specificities. Finally, we introduce a strategy for deeper integration and optimised performance.**

I. Introduction

A sizeable quantum computer would be able to break most public-key cryptography used today. This includes signature schemes such as RSA [1] and DSA [2]. Indeed, these schemes are vulnerable to Shor's algorithm [3], which can run on quantum computers. While no large quantum computer is available yet, engineering progress is rapid [4], [5]. Standards bodies and governmental agencies are thus scrambling to publish scrutinised specifications of the most promising *post-quantum* cryptographic schemes. Such schemes may resist quantum-computing-supported attacks. A shift of focus towards post-quantum cryptography is under way at the NSA [6], NIST [7] and IETF [8], [9].

A. Motivation

Post-quantum schemes — signature schemes in particular — are still chiefly academic and cannot yet inspire the kind of confidence that comes with extended real-world use. To build up trust in these schemes, their use in concrete scenarios must be evaluated. They then must be implemented and deployed in realistic contexts. This process ought to be initiated now, so those schemes can be deployed in time and be relied upon once large-scale quantum computers emerge. Hash-based signatures (HBS) [10] are a promising category of post-quantum signature schemes. Their theory and security is well-understood and they are currently undergoing standardisation. However, since they have not been used widely before and feature unique characteristics such as the statefulness of secret keys, a gap between theory and practice remains. One of the most important use cases regarding the Internet are TLS connections, e.g. to secure traffic via the HTTP protocol. A natural question is how HBS fit this use case. While toy models for this use case do not indicate immediate issues, wider deployment raises challenges that we will discuss. Independently of whether HBS should be used for TLS, integration in cryptographic libraries is critical for a wide range of use cases.

B. Contributions & Outline

In this paper, advancing practical use, we investigate the integration of HBS into concrete security protocols. We examine use cases such as S/MIME and TLS, and evaluate the use of HBS in these respective contexts. Independently of specific security protocols, we describe our integration of the XMSS (*eXtended Merkle Signature Scheme*) [11] HBS scheme into the OpenSSL open source security library [12], providing significant potential for reuse. This integration includes aspects such as OpenSSL's generic EVP (*EnVeloPe*) abstraction layer, ASN.1 (*Abstract Syntax Notation One*) [13] and X.509 formats, used in a wide range of applications. Since OpenSSL remains sparingly documented, the description of our modifications itself can be profitably reused as a guide to integrating new primitives into the library.

The remainder of this paper is organised as follows. We first recall basic features and specificities of HBS (Sec. I-C). We then examine different use cases in the context of HBS. We start with the integration of XMSS in OpenSSL, via the high-level EVP API, ASN.1 encodings and X.509 formats (Sec. II). We next implement new cipher suites for TLS using XMSS for authentication. XMSS-based email authentication with S/MIME is then described (Sec. III), completing the OpenSSL modifications performed previously with additional ones pertaining to the CMS (*Cryptographic Message Syntax*) format. Related work, conclusions and future work follow (Sec. IV).

C. Characteristics of Hash-Based Signature Schemes

Due to space constraints, we only recall functional aspects of HBS here. A full treatment of HBS can be found in a survey by Buchmann et al [10].

Unlike common signature schemes like RSA or DSA, HBS do not rely on the hardness of mathematical problems. Only the security of the underlying hash function is relied upon. Security assumptions for HBS are thus minimal.

Most HBS schemes are — inconveniently — stateful. This is not the case of the recent SPHINCS [14] scheme, which relies on the use of a few-time signature scheme: instead of yielding a complete break in security as in one-time signature schemes, multiple signing with a given key pair results in a progressive shrinking of security. The other major difference with stateful schemes is that indexes are selected randomly rather than sequentially. On the downside, signature sizes are significantly higher than for stateful schemes, and SPHINCS has not yet benefited from the extensive scrutiny that XMSS undergoes as part of its ongoing standardisation process. Summing up, the two main issues with most HBS schemes are the statefulness of the private key and the fact that an upper bound on the maximal number of messages to be signed with a given secret key exists. McGrew et al. [15] recently showed how statefulness can be managed in practice for non-distributed scenarios. For instance, the *state reservation* approach consists in storing a secret key that corresponds to a future state, thus reserving an arbitrary number of signatures for use. This avoids the need to write the updated private key into nonvolatile storage after every single signature. When generating a key, the issuer of the key has to clarify which security level is needed, what performance has to be achieved and how many signatures have to be written in the concrete context. A mechanism to exchange an exhausted key with a new one is required. Unfortunately, HBS do not fit common cryptographic interfaces. Nonetheless, their advantages are considerable and their deployment is feasible, as we will show next.

II. Integrating XMSS with OpenSSL and TLS

This section addresses the integration of XMSS into OpenSSL, with TLS as an application example. We point out the benefits of cryptographic library integration, describe the specificities of OpenSSL, and evaluate our approach by detailing the integration we performed. We integrate our reference implementation [16] — which supports XMSS and XMSS^{MT} — with both the EVP and ASN.1 layers of OpenSSL, making a full XMSS-authentication-based TLS session possible. As a side effect, we also provide S/MIME support for email authentication; this is deferred to Sec. III.

A. OpenSSL Organisation, Abstraction Layers and Formats

OpenSSL consists of two main library components. Its `libcrypto` part features implementations of cryptographic schemes, and corresponding X.509 support. Its `libssl` part implements TLS security protocols, relying on `libcrypto` for the selected cryptographic algorithms. As a general purpose toolkit, OpenSSL also offers command line tools for key parameter creation, X.509 certificate handling, S/MIME operations and so on. The EVP abstraction layer is specific to OpenSSL, and consists of high-level cryptographic interfaces to all asymmetric algorithms. Both generic and scheme-specific functions are included for ASN.1, a standardised formal notation for networking data description interoperability. Internally, OpenSSL uses numbered values called NIDs (*Numeric IDentifiers*). NIDs are used to identify cryptographic primitives, sometimes with specific parameter sets. They are also used for ASN.1 OIDs (*Object IDentifiers*). As is standard for TLS, cipher suites are defined. Cipher suites are named combinations of algorithms used for client/server negotiation, notably for authentication and encryption. Fig. 1 shows an overview of the components relevant to our contribution.

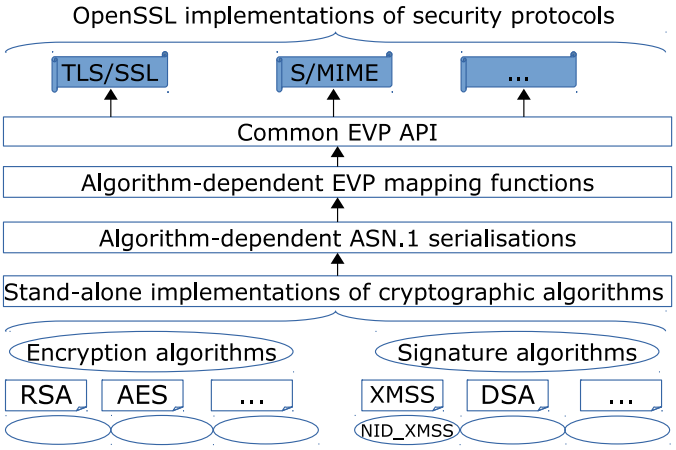

Fig. 1. OpenSSL formats and layers: stand-alone implementations of cryptographic algorithms are identified by numeric identifiers (NIDs). Their signature and/or key structures are specified in ASN.1 independently. Bridge functions are defined to map algorithm-dependent functions, such as signing methods, to the common EVP API. Security protocols are then implemented largely independently of encryption and signature algorithms.

B. EVP Integration

In OpenSSL, EVP — or envelope — functions constitute a high-level interface to cryptographic functions. They provide a consistent API that does not depend on the underlying algorithm. As a result, integrating a new cryptographic scheme into OpenSSL requires providing a translation between the stand-alone implementation of the scheme and the generic EVP functions. In practice, this is done by writing new functions that link existing stand-alone scheme functions to these generic ones, which should not be modified. The benefit of integrating with the existing EVP framework is that higher-level mechanisms, such as security protocol implementations that work with EVP (e.g. TLS), do not require substantial modification if the integration is done properly. This is because these mechanisms do not refer to specific schemes, but can be instantiated, instead, with any scheme fulfilling the necessary requirements and equipped with an EVP API.

We therefore defined new functionality on the EVP level by defining a static, XMSS-specific `EVP_PKEY_METHOD` (an array of function pointers) that can be provided to OpenSSL at runtime. The functions defined in this `EVP_PKEY_METHOD` for XMSS are then implemented, providing a bridge between already existing reference implementation functions and the EVP layer. The arguments of these functions must follow a pre-established pattern, defined in `crypto/evp/pmeth_fn.c`. For this reason, sometimes not all of them are actually used in the function body. As an example, consider `pkey_xmss_verify`, used for signature verification (Listing 1). Here, `siglen` is declared as an argument — even though it is not actually needed in the internal signature verification function — to fit the generic template of a `pkey_xmss_verify` function.

```
int pkey_xmss_verify(EVP_PKEY_CTX *ctx, const
    unsigned char *sig, size_t siglen, const
    unsigned char *tbs, size_t tbslen)
{
  (void)(siglen);
  XMSS_PKEY_CTX *xmss = (XMSS_PKEY_CTX*) ctx->
      pkey->pkey.ptr;
  unsigned char *pk_all = OPENSSL_malloc(xmss->
      para->n * 2);
  xmssPKtoStr(pk_all, xmss->pk, xmss->para->n);
  int ver = xmss_verification(pk_all, (unsigned
      char*)sig, (unsigned char*)tbs, tbslen,
      xmss->para, xmss->h);
  free(pk_all);
  return !ver;
}
```

Listing 1. EVP integration — method mapping

C. ASN.1 Serialisation

Abstract Syntax Notation One (ASN.1) is a formal notation to describe networking data; it is used for serialisation. ASN.1 is standardised by the International Telecommunication Union Telecommunication Standardization Sector (ITU-T) [13]. It provides consistency across systems, hiding platform-dependent specificities, and is associated with standardised encoding rules, i.e. the Distinguished Encoding Rules (DER). The fact that DER yields unequivocal encodings is critical for some cryptographic applications. Since ASN.1 is for instance commonly used in X.509 certificates, it is supported by OpenSSL and required to handle such certificates.

To achieve integration between the XMSS signature scheme and this notation, we first defined serialised structures. For instance, the structure for an XMSS secret key is defined as `struct xmssasn1sk`, as shown in Listing 2.

An `ASN1_SEQUENCE` is then defined, building upon this `xmssasn1sk` structure. We next invoke nested OpenSSL macros like `IMPLEMENT_ASN1_FUNCTIONS` to automatically generate helper functions based on the definitions of the `ASN1_SEQUENCE`. Encodings and decoding algorithms are implemented as words for a state machine. These macros are defined in `crypto/asn1/asn1t.h` and expand to define numerous functions relevant to ASN.1 handling, including functions for DER encoding and decoding.

```
typedef struct {
  long idx; long n;
  wots_param *param; long h;
  ASN1_OCTET_STRING *PRF;
  ASN1_OCTET_STRING *root;
  ASN1_OCTET_STRING *seed;
  ASN1_OCTET_STRING *nodes;
  ASN1_OCTET_STRING *wots_keys;
  ASN1_OCTET_STRING *adrs;} xmssa1sk;
```

Listing 2. ASN.1 integration — XMSS secret key serialisation

D. NID Assignment

NID values are used in OpenSSL protocol implementations, such as TLS, in conditional statements used to determine selected cryptographic algorithms. OpenSSL features an internal NID table. New NID values can be assigned a priori or at runtime. A priori assignment is performed in `crypto/objects/objects.txt` and related macro definitions in the same folder. Runtime assignment can be done by calling `OBJ_create` with an OID as first argument. We opted for runtime assignment using a temporary OID, since the official OIDs for XMSS parameter sets will only be assigned once the current Internet-Draft [9] becomes an RFC.

E. Cipher Suite Extension

To support TLS security protocols using XMSS for authentication, we defined new OpenSSL cipher suites. An existing cipher suite featuring Diffie–Hellman key exchange was modified to use XMSS instead of pre-quantum server authentication. Since the Diffie-Hellman problem can be solved efficiently if the discrete logarithm problem can be solved, the Diffie-Hellman key exchange protocol is not quantum-safe, unlike XMSS. It would be suitable to implement a cipher suite featuring a post-quantum key exchange protocol, so as to obtain a fully post-quantum cipher suite. In practice, cipher suites are defined in the `libssl` part of OpenSSL, in the file `ssl/s3_lib.c`. They are specified in one large array, `OPENSSL_GLOBAL SSL_CIPHER ssl3_ciphers[]`. Our new suite was defined as shown in Listing 3.

```
{ 1,
  "DHE-XMSS-AES256-GCM-SHA384",
  TLS1_CK_XMSS_WITH_AES_256_GCM_SHA384,
  SSL_kEDH, // Diffie-Hellman key exchange
  NID_xmss, // XMSS signing
  SSL_AES256GCM,
  SSL_AEAD,
  SSL_TLSV1_2,
  SSL_NOT_EXP | SSL_HIGH | SSL_FIPS,
  SSL_HANDSHAKE_MAC_SHA384 | TLS1_PRF_SHA384,
  256,
  256,},
```

Listing 3. Cipher suite declaration with XMSS authentication. OpenSSL cipher suites are represented internally as NID lists.

F. Evaluation

We implemented functionality tests to evaluate all modifications described so far. PEM (a container file

format) file creation utilities were defined for XMSS secret keys, public keys (using an XMSS secret key as an input), and certificates (using both keys as inputs). We used self-signed certificates to simplify the prototype setup. A certificate verification utility was also implemented. The selected format consists in base64 translations of X.509 ASN.1 keys. Keys are generated by the relevant common API EVP functions, which call XMSS-dependent EVP interface functions under the hood because the XMSS NID is set for the active EVP context. Once the `EVP_PKEY`[1] object is generated, it is written to disk using a generic OpenSSL PEM routine, e.g. `PEM_write_PrivateKey` for the XMSS private key. Furthermore, we implemented two tools `ssl_client` and `ssl_server` to communicate with each other, and to perform a full TLS session using a chosen cipher suite. A specific cipher suite can be selected via `SSL_CTX_set_cipher_list`, with a list containing only the desired cipher suite as the second argument and the active SSL context as the first one. Our setup is outlined in Fig. 2.

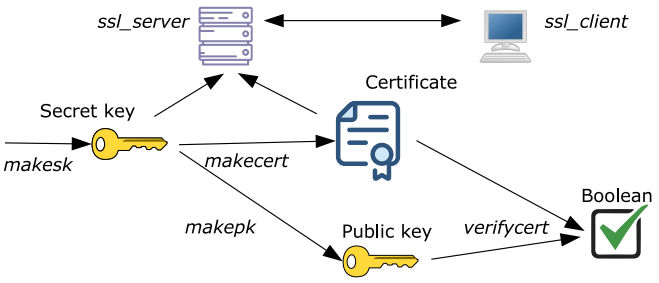

Fig. 2. Our test setup for TLS in OpenSSL, using XMSS for authentication. The tools we implemented for the tests are displayed in italics.

The tests were then instantiated using cipher suites with XMSS for authentication. As is often the case for TLS, only the server authenticated itself to the client. The fact that the desired cipher suite is actually used in the resulting TLS session can be verified using a network analysis tool. Fig. 3 shows one of our experimental protocol runs as displayed by the Wireshark packet analyser [17].

Since Wireshark only displays cipher suite names for official OpenSSL suites, our custom-made cipher suite using XMSS for authentication can only be displayed using a numerical identifier, `0xd009` in this example. At 2750 bytes, the resulting certificate is larger than typical classic ones using RSA or DSA, but not excessively so. A short tree height h was used here, but higher trees do not yield much larger signatures since signature size is linear in h for plain XMSS.

G. Practical Considerations for TLS

TLS has many applications. HTTP over TLS (HTTPS) is the most widespread and we focus on it to narrow down our analysis. An important requirement for HTTPS is the possibility of high-volume signing, since a new signature is required for every connection without session resumption. This entails fast signing. Unoptimised signature generation for HBS is very slow (hundreds of milliseconds, or worse), mainly because of the computation of the authentication path. As mentioned before, optimised algorithms for authentication path computation exist. The resulting speed-up is remarkable, about two orders of magnitude. Both XMSS and XMSS^{MT} have been shown to yield signature time under 10 ms for some parameter sets [11], [18]. This is a range comparable to RSA signing.

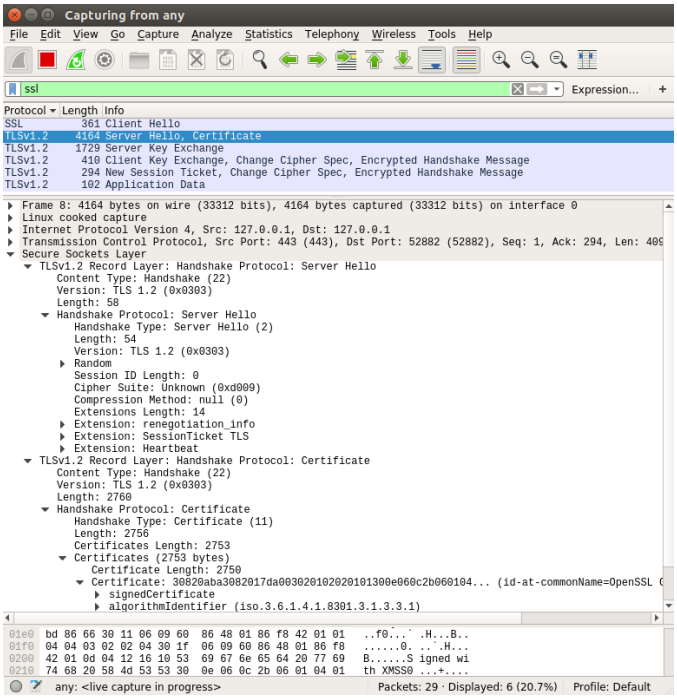

Fig. 3. A TLS protocol run using XMSS for authentication, as shown in Wireshark.

Signature size is not an issue for HTTPS, since the average size of a web page is already over 2 megabytes as of 2017 [19]. As a consequence, parameter sets prioritising signing speed over signature size seem desirable for HTTPS. This translates to using XMSS^{MT} [18], the multi-tree variant of XMSS, with a high number of layers (parameter `d` in the IETF specification [9]). More layers yield larger signatures but decrease signing time. To prevent loss of performance due to state synchronisation (I/O), using a state reservation strategy [15] is essential. Computing a future state entails setting the key index to a higher value in advance. The signing process consuming the reserved signatures requires an internal counter to keep track. The computation of the future state may be resource-intensive, possibly delaying signing if the timing is unfortunate. Because of the aforementioned high-volume signing inherent to TLS, state management is especially complex for this use-case and may even cause timeouts. Worse, multiple signing processes may be used in practice for faster signing.

[1]This type is used for both public and private keys in OpenSSL.

This makes state management even more complicated.

In practice, the limit on the number of messages to be signed must be balanced with performance. Real-world scenarios feature not one but many servers, all signing on behalf of a single logical entity. To tie all of them to a single global public key, at least two options exist. They both boil down to using XMSS^{MT}, either explicitly or implicitly:

- A classic PKI approach, where the logical entity issues short-lived X.509 certificates authenticating the signing servers (intermediate Certification Authority). Short-lived certificates are used to limit the number of signatures generated per XMSS secret key. By relaxing this constraint, XMSS parameter sets with shorter trees (e.g. smaller values of h in the specification) can be used, offering better performance. The logical entity signs the certificates using (plain) XMSS to prioritise size over signing speed. Indeed, certificate signing speed has no impact on the actual TLS sessions experienced by HTTPS clients. Signing servers use XMSS^{MT} with numerous layers to prioritise speed;
- A single XMSS^{MT} key can be divided among signing servers such that each controls a unique set of one-time-signature keys. The structure of XMSS^{MT} lends itself to this approach, since its bottom level comprises subtrees that can be partitioned neatly. However, this introduces complexity that is otherwise taken care of by the PKI approach. This approach is only superior if integration with an existing PKI is impossible.

A CA-revocation-tolerant PKI with XMSS is discussed in detail by Braun [20].

In particular for PKI, inertia of existing processes may be an obstacle to the adoption of HBS. A more flexible PKI architecture [21] could ease the transition to HBS, but does not appear to be under active consideration by stakeholders. Summing up, performance constraints do not seem problematic for this use case, but the necessary process and architecture modifications are challenging.

H. A Strategy for Deeper Integration

A fully post-quantum cipher suite implementation is desirable. To this end, we are currently integrating our OpenSSL fork with the Open Quantum Safe (OQS) project [22], a fork of OpenSSL 1.0.2 that adds ring learning with errors post-quantum key exchange support, including for cipher suites.

An outstanding issue is the handling of stateful secret keys on the EVP level. While the state is updated correctly at runtime in our implementation, it is not persisted to nonvolatile storage. Persisting the state is security-critical because a loss of state may lead to multiple use of a one-time signing key. The way in which the state is persisted is implementation-dependent; some strategies are described in [15]. At any rate, persisting the state would require modifications to some primitive-independent EVP functions, as they assume secret keys to be stateless. To solve this, we plan to add a flag specifying statelessness versus statefulness to `EVP_PKEY_METHOD` Newer versions of OpenSSL provide easier access to the EVP interface via setter functions. One would thus only need to specify stateful schemes as such explicitly, and modify the generic EVP signing function to persist stateful keys *during* signature generation. No change would be required for existing (stateless) schemes.

III. Secure Email: XMSS-based S/MIME

Another common use case requiring authentication is email, as supported by S/MIME. Performance requirements are as follows. Key generation time is not crucial, while the time constraints for signature generation and verification are very lax in comparison to HTTPS. A plain XMSS parameter set with $h = 20$, e.g. `XMSS_SHA2-256_W16_H20`, is appropriate for a single user. Indeed, 2^{20} signatures are enough to last more than 50 years, assuming 50 signed e-mails every single day. The rationale for using XMSS rather than XMSS^{MT} in this use case is to prioritise signature size over other performance aspects. A small signature is desirable here, since an average email is less than 100 kilobytes; so the relative expansion is much more significant than for HTTPS.

In practice, S/MIME signing relies on the Cryptographic Message Syntax (CMS) format. Since CMS values are created using ASN.1, the ASN.1 encodings we defined for HBS with TLS as an end goal can be reused. OpenSSL supports S/MIME, with RSA normally used as the signing algorithm. In summary, we implemented HBS support for S/MIME in OpenSSL by combining the following building blocks:

- ASN.1 encodings for XMSS structures, as defined for the TLS prototype;
- New CMS manipulation functions, based on RSA ones;
- New `ameth` functions such as `xmss_pkey_ctrl`, providing the missing link between the CMS layer and the EVP/ASN.1 one. These functions were then referenced in a modified `EVP_PKEY_ASN1_METHOD` structure, loaded at runtime to make them available.

In the original RSA versions of the CMS functions, different scenarios are taken into account depending on padding considerations. Since padding is not required for XMSS, this part of the high-level CMS functions is no longer required.

To evaluate our approach, we implemented two tools `cms_sign_xmss` and `cms_ver_xmss`, based on CMS demonstration functions available in `demos/cms`. For signing, an XMSS certificate and the corresponding private key are read as input. Verification is performed on the basis of the certificate and the message to be verified, which contains both the initial message and its XMSS signature. A function trace shows that many of the EVP-level functions already defined for the TLS prototype, such as `pkey_xmss_sign_init`, are being reused. This is due to the fact that both the key and the certificate are handled as in the TLS case. Only the high-level functions differ.

IV. Conclusions

In this paper, we have shown how to bring post-quantum authentication closer to practice by providing a prototypical integration of the XMSS HBS scheme in OpenSSL. On top of this cryptographic implementation, we implemented support for TLS and S/MIME in OpenSSL. In practice, we saw that state management and far-reaching architectural modifications are especially complicated for TLS, due to the tension between fast signing and timely state synchronisation. We have shown that the use case of S/MIME is much more amenable to HBS. This is mainly due to the reduced scope of required architectural changes and to their looser performance requirements, which entail simpler state management.

A. Related Work

HBS have not been commercially deployed for authentication so far. However, integration into the Bouncy Castle cryptographic library, including partial ASN.1 support, has recently been performed both for the stateful HBS scheme XMSS and for the stateless HBS scheme SPHINCS [23]. These integrations are stand-alone in the sense that only the schemes themselves are made available — they are not integrated in actual security protocols.

B. Future Work

We intend to improve our underlying XMSS reference implementation [16], adding pseudorandom key generation for reduced storage requirements and optimised authentication path computation for faster signing. Authentication path computation has a decisive impact on signing time, and can be sped up significantly by using an improved algorithm due to Buchmann et al [24]. This will permit a meaningful performance comparison with e.g. RSA. Most of the current OpenSSL integration code will remain intact with regard to these optimisations. An exception is the ASN.1 serialisation of secret keys, which will need to be modified since only a seed value needs to be stored when using pseudorandom key generation. Furthermore, real-world use in productive environments is needed to build up experience with HBS deployment. To this end, we are currently instantiating a software update authentication system using a LibreSSL implementation based on the OpenSSL integration described in this paper.

Acknowledgment

This research was supported by DFG grant BU 630/28-1.

References

[1] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, 1978.

[2] T. ElGamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," in *CRYPTO'84*, ser. LNCS. Springer, 1985, vol. 196, pp. 10–18.

[3] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM J. Comp.*, vol. 26, no. 5, pp. 1484–1509, 1997.

[4] D. M. Zajac, T. M. Hazard, X. Mi, E. Nielsen, and J. R. Petta, "Scalable Gate Architecture for a One-Dimensional Array of Semiconductor Spin Qubits," *Physical Review Applied*, vol. 6, p. 054013, 2016.

[5] R. Maurand, X. Jehl, D. Kotekar-Patil, A. Corna, H. Bohuslavskyi, R. Laviéville, L. Hutin, S. Barraud, M. Vinet, M. Sanquer, and S. De Franceschi, "A CMOS silicon spin qubit," *Nature Communications*, vol. 7, p. 13575, 2016.

[6] Information Assurance Directorate at the National Security Agency, "Commercial National Security Algorithm Suite," https://www.iad.gov/iad/programs/iad-initiatives/cnsa-suite.cfm, 2015, Accessed 2017-07-14.

[7] NIST Computer Security Division, "Post-Quantum Cryptography Project," http://csrc.nist.gov/groups/ST/post-quantum-crypto/, 2016, Accessed 2017-07-14.

[8] D. McGrew, M. Curcio, and S. Fluhrer, "Hash-Based Signatures," https://datatracker.ietf.org/doc/draft-mcgrew-hash-sigs, 2017, Internet-Draft. Accessed 2017-07-14.

[9] A. Hülsing, D. Butin, S.-L. Gazdag, and A. Mohaisen, "XMSS: Extended Hash-Based Signatures," https://datatracker.ietf.org/doc/draft-irtf-cfrg-xmss-hash-based-signatures, 2017, IETF Internet-Draft. Accessed 2017-07-14.

[10] J. Buchmann, E. Dahmen, and M. Szydlo, "Hash-based Digital Signature Schemes," in *Post-Quantum Cryptography*, D. J. Bernstein, J. Buchmann, and E. Dahmen, Eds. Springer, 2009, pp. 35–93.

[11] J. Buchmann, E. Dahmen, and A. Hülsing, "XMSS — A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions," in *PQCrypto*, ser. LNCS, vol. 7071. Springer, 2011, pp. 117–129.

[12] The OpenSSL Project, "OpenSSL: The open source toolkit for SSL/TLS," 2017, https://www.openssl.org Accessed 2017-07-14.

[13] International Telecommunication Union Telecommunication Standardization Sector (ITU-T), "Abstract Syntax Notation One (ASN.1) Recommendations," http://www.itu.int/ITU-T/studygroups/com17/languages, 2011, Accessed 2017-07-14.

[14] D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn, "SPHINCS: Practical Stateless Hash-Based Signatures," in *EUROCRYPT 2015*, ser. LNCS, vol. 9056. Springer, 2015, pp. 368–397.

[15] D. A. McGrew, P. Kampanakis, S. R. Fluhrer, S.-L. Gazdag, D. Butin, and J. A. Buchmann, "State Management for Hash-Based Signatures," in *SSR*, ser. LNCS, vol. 10074. Springer, 2016, pp. 244–260.

[16] S.-L. Gazdag and D. Butin, "Reference implementation for the Internet-Draft *XMSS: Extended Hash-Based Signatures*," http://www.square-up.org/index/publications.html#code, 2017, Accessed 2017-07-14.

[17] Wireshark Foundation, "Wireshark · Go Deep," http://www.wireshark.org, 2017, Accessed 2017-07-14.

[18] A. Hülsing, L. Rausch, and J. Buchmann, "Optimal Parameters for XMSS^{MT}," in *MoCrySEn*, ser. LNCS, vol. 8128. Springer, 2013, pp. 194–208.

[19] "HTTP Archive — Interesting Stats," 2017, http://httparchive.org/interesting.php Accessed 2017-07-14.

[20] J. Braun, "Maintaining Security and Trust in Large Scale Public Key Infrastructures," Ph.D. dissertation, Technische Universität, Darmstadt, 2015, http://tuprints.ulb.tu-darmstadt.de/4566/ Accessed 2017-07-14.

[21] M. Hartmann and S. Maseberg, "Replacement of Components in Public Key Infrastructures," in *IECON '01*, vol. 3. IEEE, 2001, pp. 2012–2016.

[22] D. Stebila and M. Mosca, "Post-Quantum Key Exchange for the Internet and the Open Quantum Safe Project," Cryptology ePrint Archive, Report 2016/1017, 2016, http://eprint.iacr.org/2016/1017. Accessed 2017-07-14.

[23] "Legion of the Bouncy Castle," https://www.bouncycastle.org/, 2017, Accessed 2017-07-14.

[24] J. Buchmann, E. Dahmen, and M. Schneider, "Merkle Tree Traversal Revisited," in *PQCrypto*, ser. LNCS, vol. 5299. Springer, 2008, pp. 63–78.