How to Make Operating Systems for Smart Cards Open[⋆]

Przemysław Błaśkiewicz, Przemysław Kubiak[⋆⋆], and Mirosław Kutyłowski

Wrocław University of Technology

Abstract. Nowadays the world of smart cards aimed for the market of ID-cards and e-administration services is divided (roughly) into two main categories: native cards and java cards. The first choice imposes lower requirements to the chip area and its energy consumption (which is especially important in case of contactless cards), whereas the second choice allows third parties to develop applications for the card, making the product less dependent on the developer of its operating system (OS). This dependency may be a crucial factor for administration of a country planning to deploy a new ID-card, for example. However, apart from increased requirements towards hardware, java cards do not provide access to low level libraries, which affects flexibility and development of new applications.

In this paper we propose a new business model for smart cards' OS developers, namely: "earn on the service" instead of "earn on the product". Applying the signature scheme described below, OS developers will have interest in publishing at least a partial specification of internal, low level API of their native OS.

On the algorithmic side, the main, two layered structure used by the signature scheme may be discerned as an extension of CMSS and GMSS variants of Merkle Signatures Scheme, and outperforms them in terms of speed of the structure (re)initialization and the number of signatures available with a single structure.

Keywords: smart card, Merkle signatures, mediated signatures

1 Introduction

1.1 Functionality

Consider the following requirements: we would like to enable the smart card's operating system to check signatures under software loaded on the card with a public key stored on the card. With possibly long lifetime of such a card and with card issuance spread over time, we wish to assure little (if any) degradation in system's security over considerable period of time. At the same time, we would like the protocol to be very simple in terms of embedded hardware usage: we shall use only ROM memory to store long term data (in particular the public key for signature verification shall not be changed during the hardware's existence). At the same time we would like to preserve infrastructure's ability to replace signature private key should it be compromised. Additionally, we wish to design the system in such a way as to allow implementation of business models concerning certification and authenticated dissemination of end-user software.

[⋆] The paper is partially supported by Foundation for Polish Science, MISTRZ project

[⋆⋆] Contact author: przemyslaw.kubiak@pwr.wroc.pl, Wrocław University of Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland

1.2 The Tool and Our Contribution

Long term cryptanalytic security is usually attributed to Merkle Signature Scheme (MSS), and requirements for key length in MSS should rather not grow in the foreseeable future. However, the notion of cryptanalytic security does not capture the risk of a private key being stolen or leaked e.g., by a fault injection attack. In case of such a compromise the corresponding public key should be revoked. But the latter would be a problem for embedded devices using that public key for input verification purposes. Limited area of the chip makes constraints for some extra functionalities of the device, like for example execution of the OCSP protocol.

On the other hand, in [1] a mediated variant of Merkle Signature Scheme has been proposed (we call the variant mMSS for short). At the same time, paper [1] completely neglects possible impact of mMSS on longevity of the public verification key and for the public key infrastructure responsible for that key. This paper develops mMSS into a protocol that fulfills the above requirements.

As an alternative to mMMS one may consider the framework depicted in Extended Access Control (EAC) standard [2]. That is, the issuer of e-ID cards places the public verification key on the card, and this key is used to authenticate terminals (and their access rights) the smart card is used to connect to. The key placed on the card is called a *trust point*. Thus the trust point could be used to additionally authenticate software loaded on the card. However, at least two factors make this approach unattractive for the manufacturer:

- The trust point needs to be updated periodically, hence cannot be embossed in ROM of the card. Moreover, the card must check whether the currently stored trust point is fresh enough to authenticate the terminal (or software), which creates additional footprint for the auxiliary management procedures on the card.
- EAC uses asymmetric public key algorithms, thus they have to be implemented on the card.

Although smart cards developed for e-ID market may have both futures already implemented, for other application areas at least one of the factors may be an obstacle. For example banking cards use EMV standard instead of EAC, and SIM cards rather utilise symmetric algorithms.

Therefore, from the OS developer's point of view different market sectors may provide different environments, and a single, low demanding tool well suited even for environments with a poor PKI infrastructure seems to be superior over a few specialized ones. The tool, once implemented and scrutinized may be embossed in ROM of cards developed for different market sectors.

1.3 Business Model

We assume that the smart card operating system manufacturer has an agreement with a producer of the microcontrollers for which the OS is designed[1] (the agreement should

[1] Note that smart card's OS highly depends on the hardware, hence publication of OS's internals is in interest of this hardware producer.

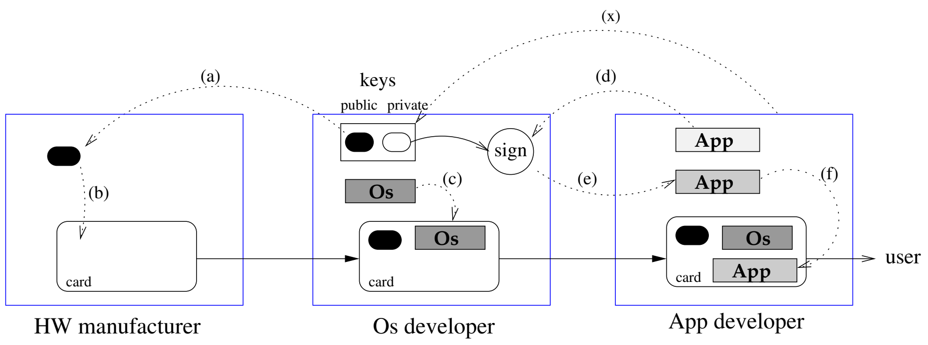

Fig. 1. Flow of events for application development. Public key presented to hardware producer (a) is embossed in card's ROM (b); OS is implemented on the card (c); application designer requests its application be verified (d) and receives signed version (e); the application is uploaded to the card (f). Operation labelled (x) depends on the model (see text).

be exclusive if the whole smart card's OS is open source licensed). In Fig. 1 the flow of events in the process of developing a smart-card application is presented. First, the OS manufacturer presents a public key to the hardware manufacturer ((a) in Fig. 1), which is then embossed in ROM of the microcontroller (b). *The payment for the producer is roughly proportional to the number of microcontrollers having the same public key embossed.*

Considering the key infrastructure, three cases need to be distinguished. This is marked as action (x) in the figure – indicating the source of public and private keys pair.

- The contracting party (e.g., an issuer of e-ID cards) that purchases microcontrollers (possibly with the kernel of the operating system installed on them[2]) through the OS developer is able to maintain the signature service on its own (as is the case with government structures, for example). Therefore the verification key chosen by the contracting party is passed to the producer of the microcontroller. Action (x) takes place, but both (d) and (e) are performed *internally* by the application developers cooperating with the contracting party, which generates the signatures.
- The contracting party purchases not only the microcontrollers (with the core of the OS on them), but also pays for each signature (actions (d) and (e)) made for the series of cards purchased (cf. Subsect. 1.4). Hence this is the OS producer who serves the signature service. Accordingly, the public key chosen by the OS producer is embossed in the microcontrollers. There is no (x) action. *Initial payment may be with a discount. However, the OS manufacturer will additionally earn on the signature service. This option may be attractive for banks and private companies.*
- The OS manufacturer may itself purchase some number of microcontrollers, and sell them directly to end users, who may enjoy developing new open source applications (e.g., students or hobbyists), or may need a secure platform for some

[2] If the kernel is not on open source license then it must be protected from reading by user-applications.

specialized software developed by someone else (the software developer may be of open source or of commercial type). The OS manufacturer will charge the end-user for each signature made exclusively for the user's card (cf. Subsect. 1.4). Action (x) does not happen, (d) and (e) are executed internally by the OS developer.

Each series sold for an institutional user is differentiated by the public key embossed (and thus by the trees from Fig. 2 on page 7 constructed during a series of signatures created). Smart cards sold by the OS manufacturer directly to the end-users may all store the same public key.

1.4 Responsibility

Two factors should be established when software is signed by the OS manufacturer (actions (d) and (e)):

- the set of cards the software may be loaded on,
- who is responsible for possible damages the software could make on the cards.

For this purpose we propose that each file before being signed is prepended with a number of control fields. The first field carries either the serial number of the microcontroller the file is addressed for, or the "broadcast" number, which means that the file may be uploaded to any microcontroller which has the verification key corresponding to the signature key used to sign the file.

The next field indicates the number of the agreement (or the license), which points to party responsible for the signed software. If responsibility is on the side of the OS manufacturer, then it should have it certified or perform certification itself. The latter case may of course be reflected in the price for the signature.

After successful verification of the addressee number and the signature, the control fields are removed by the microcontroller from the file loaded, and the file is internally marked as valid. The signature shall not be stored on the card after its verification.

1.5 Related Work in Application Area

Apart from EAC mentioned in Subsect. 1.2, we leverage our idea from already existing products and solutions. Perhaps the most prominent is that of **Nokia**'s third-party application development chain [3]. In order to develop applications on Nokia's platform, a developer first needs to apply for a kind of access code corresponding to his particular phone, which enables it to accept foreign code. Next, when the application is ready it undergoes a verification and certification process by Nokia and is made available to the public. Only such application, bearing adequate signatures is accepted by all Nokia platforms.

While this application focuses hardware-level access, a slightly more high-end example is that of Windows operating systems when accepting third-party drivers and software to be installed on a computer. During installation, the software is checked to bear a signature of one of the developers trusted by the OS provider. However, if no signature is present, the user can still opt to continue the installation.

Lastly, we mention the **Java Verified** programme, which basically allows applications written in JavaME to be tested for conformity with security standards and certified. The devices used for developing the application need to be preloaded with UTI root certificate to enable them to verify R&D signature under the developed software, and then to run the code. While this is similar to the Nokia case described above, the PKI infrastructure is at full use here.

It is clear that such mechanism of authorizing code to be run on a given hardware is of vital importance. We propose a solution that relies on inexpensive symmetric cryptography and does not require the Public Key Infrastructure with all its mechanisms to be maintained by either the hardware provider or OS designer.

Other Applications of the Signature Scheme Such a low demanding tool may also be utilized to authenticate the FPGA bitstream (see [4] for a good argumentation thereof), to authenticate upgrades of software developed for cars (e.g., ignition mappings), updates of software for medical equipment or for avionics of an airplane. The latter two are examples of highly demanding areas in terms of security, yet it still remains conceivable that responsibility for secure execution of software is shifted to parties other than hardware or OS developers.

Other area is hardware used in e-voting, in which transparency of the development and maintenance processes is essential.

Notation. In the subsequent sections we mainly follow notation from [5]: $H : \{0,1\}^* \to \{0,1\}^\ell$ for a hash function used in one-time signatures, $\tilde{H} : \{0,1\}^* \to \{0,1\}^{\ell'}$ for a hash function used to build Merkle trees, $G : \{0,1\}^* \to \{0,1\}^n$ for a hash function used to calculate messages' digests (usually it is assumed that G is collision resistant, however, in order to not entirely rely on collision resistance, message randomization trick known from Schnorr signatures may be used – see the end of Subsect. 3.2), $\oplus$ is exclusive or bit-wise operation (xor operation), $a \in_R A$ for a choice of a from set A with uniform probability distribution. In the protocol proposed it is assumed that $\ell' = n$ (hence $\tilde{H} = G$ may be used).

2 Mediated Merkle Signatures from [1]

Mediated signature scheme [6] is a two-party variant of a signature scheme (in [6] it was instantiated with RSA). The signature is valid if and only if each of the two participants has correctly executed the signature protocol. One of the participants in this two-party setting is a central server – in this way end-user's device may be quickly prevented from making a valid signature. This is especially useful when end user's device is lost, or the user is fired and is not allowed to make signatures on behalf of her/his organization/enterprise any more.

Below we apply the idea of mMSS [1] to a more symmetric setting, i.e., to the setting in which the signature key is composed of subkeys in possession of separate servers (or rather Hardware Security Modules) having similar capabilities. Thus we are interested in a classic, multi-party setting.

Let us recall one of the two ideas from [1] for making a multiparty version of MSS. Let $\varGamma \geq 2$ denotes the number of parties (exact value of $\varGamma$ should follow from security level assured by the Hardware Security Module (HSM) and from efficiency of the microcontroller deployed on end-users' devices). Then each one-time public key $Y_j, j = 0, 1, \ldots, 2^h - 1$ in MSS (for description of MSS we refer the reader to Appendix A), that is each argument for creating the j-th leaf $\text{NODE}_{j,0} = \tilde{H}(Y_j)$, is a concatenation of $\varGamma$ one-time public keys of consecutive parties (we assume that participants of the signature scheme are indexed). That is

$$Y_j = Y_{j,1}||Y_{j,1}||\ldots||Y_{j,\varGamma}, \tag{1}$$

where $Y_{j,i} = (y_1^{(j,i)}, y_2^{(j,i)}, \ldots, y_t^{(j,i)})$ is the one-time public key of the i-th party.

Each key $Y_{j,i}$, $i = 1, 2, \ldots, \varGamma$ is distributed among parties after its generation by the i-th party. Thus each of the parties can reconstruct the leaf $\text{NODE}_{j,0} = \tilde{H}(Y_j)$ and the complete Merkle tree (nodes $\text{NODE}_{j,0} = \tilde{H}(Y_j)$ are also needed for Merkle tree traversal algorithm reconstructing authentication paths (6)). The root $\text{NODE}_{0,h}$ of the tree is the public key of the mMSS (we call it *multiparty MSS*).

2.1 Signature generation.

Assume that $s \in \{0, 1, \ldots, 2^h - 1\}$ is the index of the first unused leaf of the Merkle tree. Let M be a message to be signed. Each party $i \in \{1, 2, \ldots, \varGamma\}$ generates its own one-time signature $\sigma_{\text{OTS}}^{(i)}(G(M))$ of message digest $G(M)$. Next the signatures $\sigma_{\text{OTS}}^{(i)}(G(M))$, $i = 1, \ldots, \varGamma$ are distributed among the parties. Each of the parties verifies the one time signatures, reconstructs authentication path (6) for $\text{NODE}_{s,0}$ and verifies if the path ends on the root of the tree. The complete mMSS signature of M is:

$$\begin{aligned}(s, \sigma_{\text{OTS}}^{(1)}(G(M)), \ldots, \sigma_{\text{OTS}}^{(\varGamma)}(G(M)), Y_s,\\ (\text{NODE}_{(s/2^0)\oplus 1,0}, \ldots, \text{NODE}_{(s/2^j)\oplus 1,j}, \ldots, \text{NODE}_{(s/2^{h-1})\oplus 1,h-1})).\end{aligned} \tag{2}$$

Public key Y_s does not have to be present in the signature (2) if Winternitz-OTS [7] is used, since it is recoverable from $\sigma_{\text{OTS}}^{(i)}(G(M))$, $i = 1, \ldots, \varGamma$. However, if the improvement from [8] is used then some fragment of each $Y_{s,i}$ from formula (1) for $j = s$ must be present in place of Y_s in (2). The rest of each $Y_{s,i}$ is recoverable from $\sigma_{\text{OTS}}^{(i)}(G(M))$, $i = 1, \ldots, \varGamma$.

2.2 Signature verification.

Verification of mMSS signatures is analogous to verification of MSS ones. The only difference is verification of $\varGamma$ one-time signatures with concatenated public key Y_s (1), or reconstruction of Y_s from those $\varGamma$ one-time signatures. In the subsequent sections we assume that mMSS signature (2) may only be valid if all the one-time signatures $\sigma_{\text{OTS}}^{(1)}(G(M)), \ldots, \sigma_{\text{OTS}}^{(\varGamma)}(G(M))$ are present. However, some generalizations are possible, e.g., at least $\varGamma'$ one-time signatures must be present to start signature verification procedure, and if the OTS scheme from [8] is used *complete* subkeys $Y_{s,i}$ of key (1) must be delivered in place of lacking signatures.

3 Robust Signatures for Verifiers having Low Circuit Complexity

The protocol proposed in this paper is a kind of mixture of mMSS scheme, of the CMSS scheme [9] (its generalization is GMSS [10]) and of an idea proposed in [11, Sect.5.5]. Authors of [11] consider two solutions for extending the number of signatures available in a single Merkle tree. The first solution is obvious: the public key (i.e., the root of the tree) should be replaced with the root of the next tree. The second option is to utilize the last (i.e., the rightmost) leaf of the tree to sign the root of the next tree. Then to each signature made with the second tree the signature of its root made with the leaf of the preceding tree is attached. If all but one leaves of the second tree are used the next tree is generated and its root is signed with the last leaf of the second tree. The procedure continues.

Below we utilize the second option, but not to a single Merkle tree, but to CMSS-like structure (see Fig. 2). Moreover, to amortize cost of creation of second-level trees we propose to use an unbalanced structure – unlike in CMSS and GMSS schemes, in our scheme subtrees of the main tree have different heights.

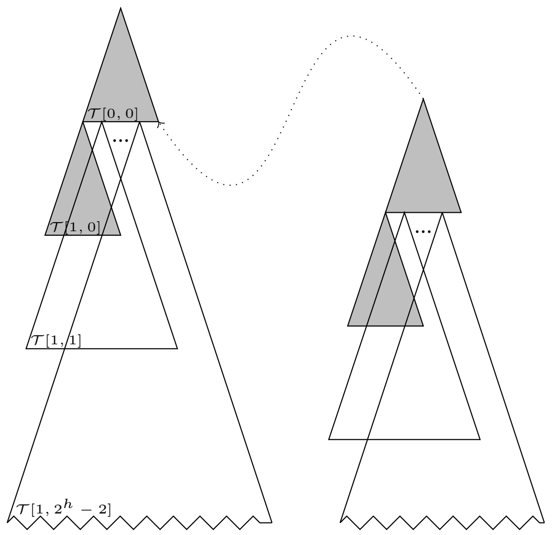

Fig. 2. Tree-hierarchy in the protocol.

3.1 Self-Certifying Signatures in a Hierarchical Construction

The signature scheme is built according to the following rules:

- Each one-time signature in the scheme is a multi-party one (hence all subtrees in the scheme are in fact mMSS trees), and all OT sub-signatures must be present to make the signature valid.
- Initially a two-layered construction is deployed (see the left-hand side structure in Fig. 2), with only two trees initialized (shown in gray). The main tree $\mathcal{T}[0,0]$ has 2^h leaves, while the leftmost subtree $\mathcal{T}[1,0]$ has $2^{h'}$ leaves. Hence, the cost of initialization of both gray trees in the structure is proportional to $2^h + 2^{h'}$.

- The subtrees signed by the leaves of the main tree have capacities of $2^{h'}$ (the gray tree $\mathcal{T}[1,0]$), $2^{h'+1}$ (tree $\mathcal{T}[1,1]$), $\ldots$, $2^{h'+2^h-2}$ (tree $\mathcal{T}[1,2^h-2]$) signatures. Place under the rightmost leaf of the tree $\mathcal{T}[0,0]$ remains empty when the structure is used up – that leaf shall be used for signing the next structure (depicted by the arrow in Fig. 2).
- Each signature from the two-level structure contains two parameters at its beginning: index s of the leaf being used in the subtree, length of the authentication path (6) in that subtree. The length cannot be greater than $h' + 2^h - 2$ and (since it is of the form $h' + j$) gives index j of the leaf of the main tree that is used to sign root of $\mathcal{T}[1,j]$.
- The subtree on the right hand side of tree $\mathcal{T}[1,j]$, that is the subtree $\mathcal{T}[1,j+1]$, for $j = 0, 1, \ldots, 2^h - 3$, may be built while the tree $\mathcal{T}[1,j]$ is being used up: when a single leaf of $\mathcal{T}[1,j]$ is used, two leaves of $\mathcal{T}[1,j+1]$ could be generated and hashed (so the tree $\mathcal{T}[1,j+1]$ is gradually built in left-to-right order. This way construction of the structure is amortized.
- When the whole leftmost structure is used up or one of the HSMs becomes compromised the next structure for $2^{h'+2^h-1} - 2^{h'}$ signatures is initialized (this structure has new private keys, that is new SEEDs – see Appendix A). The root of this new structure is signed with the rightmost leaf of the main tree $\mathcal{T}[0,0]$ of the first structure, and then private keys (the SEEDs) of the old structure are erased. No more signatures could be made with the old structure – as we see compromise of a single HSM and its private keys is not enough to compromise the scheme.

Since each one-time signature scheme is a multi-party one (cf. OT public key (1)) and all Γ one-time sub-signatures must be present to make verification of the OT signature possible, the OT signatures are in fact distributed ones. Therefore the scheme proposed does not need Certificate Authority because in this distributed setting HSMs are continually certifying each other: it suffices that not-compromised HSMs aborts execution of the protocol to prevent signing some particular code.

Due to the trick from [11, Sect.5.5] replacement of the structure does not imply replacement of the verification key (i.e., of the root of $\mathcal{T}[0,0]$): to each new signature a signature of the new root that verifies with the old root is attached, but private keys of the old structure are erased. Note that the same trick could be made with any distributed signature scheme, but Merkle signatures are supposed to be more resistant to cryptanalysis than other schemes. Moreover, arithmetic required on the side of the verifier is very simple: only hash functions are needed.

3.2 Security of the Scheme

Since 128-bit output of H is sufficient (cf. [8]) function H may be instantiated with a fast hash function built from a 128-bit block cipher operating with a 128-bit key (cf.. e.g., [12]). Colliaion resistance property is required from G and $\tilde{H}$. A good choice in this case may be e.g., a double-pipe construction of Lucks [13] (see [14, Sect.4] for strong arguments for security of the construction). Another option is to use a hash function utilizing double block length compression function based on a 128-bit block cipher using 256-bit key [15].

Still, we should take a closer look on the requirement of collision resistance of $\tilde{H}$ in our scheme. Indeed, assume that it is easy to find collisions in $\tilde{H}$. Thus to exploit weaknesses of $\tilde{H}$ an attacker needs pairs of arguments for $\tilde{H}$ having special properties or/and needs many such arguments to leverage the birthday paradox. However, in MSS the initial arguments of $\tilde{H}$ are OTS public keys, i.e., some sequences of outputs of H. Since H gives pseudorandom output we may assume that the attacker must generate some set of "candidate" OTS private keys. Accordingly, consider the following scenario: a party generating MSS public and private key pair generates a set of private OTS keys, and then a set of public OTS keys. These public keys are hashed with $\tilde{H}$ and if collision is found, the colliding OTS keys will be used in the following attack. One of the keys is used to produce some leaf of the MSS tree, and when that key is used to sign some message, the colliding key may be used to sign any other message. This "extra" signature is then correctly verified with the root of the tree. As the colliding private keys are different, then both signatures may be revealed without a risk connected with OTS: if two OTS signatures of different messages are made with the same private key, then any party can make a signature of some other (maybe a nonsense) message. The attack above can be easily extended on higher levels of nodes. The crucial point of this attack is that the adversary can in advance make any number of candidate OTS keys, and then choose the colliding ones. But *our scheme is a distributed one*, and only OTS public keys $Y_{j,i}$ (from (1)) submitted by separate HSMs are used to produce a leaf $\tilde{H}(Y_j)$. If at least one HSM behaves honestly and privately generates its own component $Y_{j,i}$ for leaf $\tilde{H}(Y_j)$ then the described collision attack immediately reduces to a second preimage attack: compromised HSMs have no time to adjust their arguments of H to produce collisions for leaf $\tilde{H}(Y_j)$, and once some arguments are chosen and $\tilde{H}(Y_j)$ is calculated then to find an OTS key for an"extra" signature a second preimage of the value of $\tilde{H}(Y_j)$ must be found, or a series of second preimages for a value of $\tilde{H}$ on some higher level. To minimize the possibility of any adjustments of arguments of H made in real time to find collisions for leaf $\tilde{H}(Y_j)$, commitments $H(Y_{j,i})$ to $Y_{j,i}$, $i = 1, 2, \dots, \Gamma$, may be submitted first by all HSMs, and then $Y_{j,i}$ would be revealed to produce value of $\tilde{H}(Y_j)$.

The collision attack on MSS described above, if ever possible, could be devised by the implementer of the device: maliciously and carefully chosen set of seeds is planted on the device to later make some "extra" signatures with colliding key(s) present outside the device. Therefore care should be taken to assure that our distributed system is distributed indeed. Preferably, the HSMs should be built by different teams.

As refers to G: collision resistance of message digest function is a common requirement in signature schemes. On the other hand, to strengthen the protocol the signature service may always prepend a random bitstring to the message before hashing and signing it. To each new signature the bitstring should be freshly and distributively generated by the HSMs with use of a cryptographically secure protocol (i.e., bit commitments). Smartcards may be designed in such a way that this initial random bitstring is removed from the message (i.e., from the program code) after signature verification. This message randomization technique is well justified [16].

3.3 Efficiency of the Scheme

For $h = 6$, $h' = 7$ each of the two-layered structures has capacity of $2^{70} - 2^7$ signatures, and for $h = 7$, $h' = 7$ theoretical capacity of $2^{134} - 2^7$ signatures. *As a side result of our construction and parameter choice we have shown that trees of height reaching values* > 60 *are feasible, despite the claim expressed in* [17]. Indeed, paper [17] estimates feasibility of key generation for tree heights up to 20, but does not take the possibility of amortization into account. Note that tree traversal algorithms like the one from [17] allow to efficiently generate authentication paths (6) even for trees of heights needed in our scheme. As a result, signing procedure executed on the side of HSMs remains efficient (SEEDs for OT private keys could be calculated as a tree structure from some root private SEEDs).

Let the scheme from [8] be used as OTS. Then to reduce storage requirements the calculation of $\text{NODE}_{s,0}$ should be done in stages by the verifier: application of compression function $\tilde{H}$ to first blocks $Y_{s,i}$ of Y_s becomes possible even before Y_s from formula (1) is completely recomputed from OTSs. Calculation cost of $G(M)$ should not be prohibitive, because due to EEPROM size constraints applications for smart cards are rather of moderate size. We quite conservatively assume that output of G has $n = 256$ bits. Then verification of a single OTS costs $\frac{1}{2} \cdot t(w-1) = \frac{1}{2} \cdot 399$ calls of H on average, where w and ℓ are parameters of scheme [8] and $w = 4$ is optimal (for $n = 256$ and for $w = 4$ we get $\ell = 133$). In a single one-time multiparty signature we have Γ one-time sub-signatures, and since the first structure is a two-level one, we need $2 \cdot \Gamma \cdot \frac{1}{2} \cdot \ell(w-1) = \Gamma \cdot 399$ calls of H on average, and $(h' + 2^h - 2 + 1) + (h + 1) = h' + h + 2^h = 77$ calls of $\tilde{H}$ for $(h, h') = (6, 7)$. Each reinitialization of the structure increases the cost of signature verification by $\frac{1}{2} \cdot \Gamma \cdot 399$ calls of H on average (counting over reinitialization events) and $h + 1 = 7$ additional calls of $\tilde{H}$.

For $\Gamma = 2$ or $\Gamma = 3$ all the values above are certainly not prohibitive[3], especially when hash functions are implemented in hardware. Note that Merkle signature verification is cheaper than signature generation (see Appendix A and Merkle-tree traversal algorithm [17]), and in case of MSS the latter was implemented on a 8-bit general purpose microcontroller [12]. What is more, users seem to be accustomed to the fact that software upgrade on a PC takes some time, hence in case of a smart card they could expect similar phenomenon.

Notably, the contribution of OTS to the final signature cost is considerable, thus the two-level unbalanced structure outperforms the multi-level GMSS solution.

Signature size is a less important factor than verification time: the signature would not be stored on the card. It would be proceeded "on the fly": consecutive iterations of compression function of $\tilde{H}$ would gradually consume Y_s (1) while Y_s would be gradually reconstructed from consecutive blocks of consecutive one-time sub-signatures. Therefore it suffices to transfer signature to a card in portions, a next portion is sent by a reader if the acknowledgement is received from the card.

[3] In fact each single call of H in the scheme could be replaced with a single call of its compression function, whereas a single call of $\tilde{H}$ in the scheme corresponds to *a few* calls of the compression function of $\tilde{H}$. The number of calls of the latter compression function depends on its compression ratio, but in any case it does not dominate in the cost of signature verification.

References

1. Kubiak, P., Kutyłowski, M.: Polish concepts for securing e-government document flow. In: ISSE 2010 Proceedings, Vieweg + Teubner Verlag (2010)
2. BSI: Advanced Security Mechanisms for Machine Readable Travel Documents 2.1. Technische Richtlinie TR-03110-2 (2012)
3. Nokia Developer: Packaging and signing. (online: `http://www.developer.nokia.com/Distribute/Packaging_and_signing.xhtml`)
4. Drimer, S.: Authentication of FPGA bitstreams: why and how. In: In Applied Reconfigurable Computing, volume 4419 of LNCS. (2007) 73–84
5. Dahmen, E., Okeya, K., Takagi, T., Vuillaume, C.: Digital signatures out of second-preimage resistant hash functions. In: PQCrypto. (2008) 109–123
6. Boneh, D., Ding, X., Tsudik, G., Wong, C.M.: A method for fast revocation of public key certificates and security capabilities. In: SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium, Berkeley, CA, USA (2001) 22–22
7. Merkle, R.C.: A certified digital signature. In: CRYPTO. (1989) 218–238
8. Buchmann, J., Dahmen, E., Ereth, S., Hülsing, A., Rückert, M.: On the security of the Winternitz one-time signature scheme. In: AFRICACRYPT. (2011) 363–378
9. Buchmann, J., García, L.C.C., Dahmen, E., Döring, M., Klintsevich, E.: CMSS - an improved Merkle signature scheme. In: INDOCRYPT. (2006) 349–363
10. Buchmann, J., Dahmen, E., Klintsevich, E., Okeya, K., Vuillaume, C.: Merkle signatures with virtually unlimited signature capacity. In: ACNS. (2007) 31–45
11. Naor, D., Shenhav, A., Wool, A.: One-time signatures revisited: Have they become practical? Cryptology ePrint Archive, Report 2005/442 (2005)
12. Rohde, S., Eisenbarth, T., Dahmen, E., Buchmann, J., Paar, C.: Fast hash-based signatures on constrained devices. In: CARDIS. (2008) 104–117
13. Lucks, S.: A failure-friendly design principle for hash functions. In: ASIACRYPT. (2005) 474–494
14. Hoch, J.J., Shamir, A.: On the strength of the concatenated hash combiner when all the hash functions are weak. In: ICALP (2). (2008) 616–630
15. Fleischmann, E., Forler, C., Lucks, S., Wenzel, J.: Weimar-DM: A highly secure double-length compression function. In: ACISP. (2012) 152–165
16. Neven, G., Smart, N., Warinschi, B.: Hash function requirements for Schnorr signatures. Journal of Mathematical Cryptology **3**(1) (2009) 69–87
17. Buchmann, J., Dahmen, E., Schneider, M.: Merkle tree traversal revisited. In: PQCrypto. (2008) 63–78
18. Merkle, R.C.: Secrecy, authentication, and public key systems. PhD thesis, Deptartment of Electrical Engineering, Stanford University (1979)

A Merkle Signature Scheme (MSS)

In [18, Sect.V] Ralph Merkle proposed a protocol that allows to verify multiple (but a limited number of) one-time signatures with a single verification key: 2^h public keys of the one-time signature scheme are put in a sequence

$$(Y_0, Y_1, \ldots, Y_{2^h-1}). \tag{3}$$

Corresponding private keys are random bitstrings, but to facilitate implementations and to avoid high storage requirements those bitstrings are usually generated from some

SEED, with usage of cryptographically secure pseudorandom number generator. Then the private key reduces to that SEED value.

Hash function H is applied separately to each public key in sequence (3) and a sequence of 2^h hashes is obtained:

$$(\mathrm{NODE}_{0,0}, \mathrm{NODE}_{1,0}, \ldots, \mathrm{NODE}_{2^h-1,0}),$$

i.e., $\mathrm{NODE}_{j,0} = \tilde{H}(Y_j)$. Next a hash tree is build from these bottom level nodes (so the nodes become leaves of the tree). The tree is build according to the rule:

$$\mathrm{NODE}_{i,j} = \tilde{H}(\mathrm{NODE}_{2i,j-1} || \mathrm{NODE}_{2i+1,j-1}), \tag{4}$$

where $i = 0, 1, \ldots, 2^{h-j} - 1$, $j = 1, 2, \ldots, h$. The root of the tree, i.e., $\mathrm{NODE}_{0,h}$, is a single verification key for 2^h one-time signatures.

A.1 Signature generation.

Suppose that $s \in \{0, 1, \ldots, 2^h - 1\}$ is the index of the first unused leaf of the Merkle tree. First, to reduce the task of signing a bitstring to signing a bitstring of a predetermined length, a message digest function G is applied to message M. Then a one-time signature $\sigma_{\mathrm{OTS}}(G(M))$ of $G(M)$ is generated – the private key corresponding to verification key Y_s is used. Next the verification key Y_s is attached to $\sigma_{\mathrm{OTS}}(G(M))$ (the last step not always is necessary, cf. [7]). Let

$$(\mathrm{NODE}_{s,0}, \mathrm{NODE}_{s/2,1}, \ldots, \mathrm{NODE}_{s/2^j,j}, \ldots, \mathrm{NODE}_{s/2^{h-1},h-1}) \tag{5}$$

be the path from $\mathrm{NODE}_{s,0} = \tilde{H}(Y_s)$ to the root of the tree, where $s/2^j$ means that s is non-cyclically shifted to the right by j bits. The root $\mathrm{NODE}_{0,h}$ is public, thus it is not included in the path (5) (see that for $s \in \{0, 1, \ldots, 2^h - 1\}$ we have $s/2^h = 0$). Note that $\mathrm{NODE}_{(s/2^j)\oplus 1,j}$ is the sibling of $\mathrm{NODE}_{s/2^j,j}$ on the path (5): the $\oplus$ operation is bit-wise, hence as a result of $(s/2^j) \oplus 1$ only the least significant bit of $s/2^j$ is flipped. The following sequence of the siblings:

$$(\mathrm{NODE}_{(s/2^0)\oplus 1,0}, \ldots, \mathrm{NODE}_{(s/2^j)\oplus 1,j}, \ldots, \mathrm{NODE}_{(s/2^{h-1})\oplus 1,h-1}). \tag{6}$$

is called *authentication path* for $\mathrm{NODE}_{s,0}$. The Merkle signature $\sigma_s(M)$ of M is

$$(s, \sigma_{\mathrm{OTS}}(G(M)), Y_s, (\mathrm{NODE}_{(s/2^0)\oplus 1,0}, \ldots, \mathrm{NODE}_{(s/2^j)\oplus 1,j}, \ldots, \mathrm{NODE}_{(s/2^{h-1})\oplus 1,h-1})).$$

A.2 Signature verification.

Given

$$(s, \sigma'_{\mathrm{OTS}}(G(M)), Y'_s, (\mathrm{NODE}'_{(s/2^0)\oplus 1,0}, \ldots, \mathrm{NODE}'_{(s/2^j)\oplus 1,j}, \ldots, \mathrm{NODE}'_{(s/2^{h-1})\oplus 1,h-1})). \tag{7}$$

signature $\sigma'_{\mathrm{OTS}}(M)$ is verified with verification key Y'_s, or in the OTS schemes [7], [8] Y'_s is reconstructed on the basis of $\sigma'_{\mathrm{OTS}}(M)$. Next genuineness of Y'_s is verified: $\mathrm{NODE}'_{s,0} = \tilde{H}(Y'_s)$ is calculated first, and then consecutive values

$$\mathrm{NODE}'_{s/2^i,i} = \tilde{H}(\mathrm{NODE}'_{\min(s/2^{i-1},(s/2^{i-1})\oplus 1),i-1} || \mathrm{NODE}'_{\max(s/2^{i-1},(s/2^{i-1})\oplus 1),i-1})$$

for $i = 1, 2, \ldots, h$ are obtained (cf. the rule (4)). The signature (7) is valid if $\mathrm{NODE}'_{s/2^h,h}$ equals the public key $\mathrm{NODE}_{0,h}$.