MAVE, NEW LIGHTWEIGHT DIGITAL SIGNATURE PROTOCOLS FOR MASSIVE VERIFICATIONS

SERGIO DEMIAN LERNER

Abstract. In this paper we propose new decentralized digital signature protocols to allow massive bulk verifications (MAVE). The protocols satisfies very tight requirements of performance and storage. MAVE requires a third party with no special trust and a broadcast medium. MAVE can be used for groups of millions of users or autonomous agents that generate thousands of signatures per second that must be verified by all users in realtime, using only modest desktop computers. It can solve diverse authenticity problems in huge networks such as sensor networks or peer-to-peer virtual currency networks.

1. Introduction

To construct a signature system for massive verifications we must optimize to achieve:

(1) Short signatures: To allow the transmission of thousands of signatures per second.

(2) Short public keys: To allow every user to validate each other signatures using a percentage of available RAM, still supporting millions of users.

(3) Fast batch verification with different senders: To allow the realtime verification of signatures.

(4) Decentralization: there should be no single point of attack to the system. Any “special” node should be replaceable and every user should be able to act as a “special” node.

The problem of verifying many signatures at once, called batch verification, has been treated extensively in the literature [12] [9] [6] [11] [7]. Nevertheless, only a subset of the standard signature schemes provide batch verification and very few provide speedups for batch verification in the case of multiple senders. There has been advances in the development of a signature schemes that can verify multiple signatures of multiple senders with a significant reduced computational cost [19], but the systems require very large public keys. Traditional digital signature schemes, such as DSA, ECSA or RSA, are built using number theoretic algorithms that are, on average, thousands times slower than standard MAC and HASH functions such as HMAC and SHA1. This gap has lead to the development of means of authentication based on hash chains or one time signatures (OTS), and by switching from digital signature algorithms (off-line) to digital signature protocols (on-line) [15].

There was also developments in mixed On-Line/Off-Line schemes [8] that allow the signing operation to be fast, but does not speed-up verification, which is the main target. All schemes derived from [8] such as Server-supported Signatures [1],

Key words and phrases. Digital signatures.

Preliminary version – April 17, 2012
SAS and SAOTS [2] have this property. Also neither these protocols achieve the requirement of decentralization, since they require a permanent third party.

Most of OTP schemes, such as Lamport [10], Merkle-Winternitz [13,14], BiBa [18], Bleichenbacher-Maurer [3–5] and Powerball [15] require very long one-time public keys. These schemes require the signer to transfer each one-time public key to the sender for each signed message. In massive continuous verifications, we use the full available bandwidth continuously, we have no unused bandwidth to pre-send the public keys, so each one-time public-key must be appended to the signature itself. Also each one time public key must be verified to be authentic, either by a Merkle tree or similar constructs.

Since the problem of realtime verification of huge amounts of signatures requires the efficiently transmission of the signature messages, both signature size, as well as verification performance, are limiting factors. ECDSA generally favors short signatures over verification time. RSA signatures are much faster to verify, although longer. Lamport signatures are fast but require still longer sizes. Any of this choices impose limitations on the maximum number of signatures a user can validate per second, both in terms of bandwidth usage, storage and CPU processing time. For example, using an average desktop computer with an average Internet connection and ECDSA with a 256 bit key, the maximum number of signatures per second that can be processed is approximately 100, and the main limitation is CPU processing time.

In this paper we propose new digital signature protocols (MAVE-2, MAVE-3 and MAVE-23) or collectively the MAVE protocol family, and we compare them to existing protocols. Table 1 show a comparison of standard signature schemes and MAVE-3 using standard implementations on an average desktop computer. MAVE-3 outperforms the fastest standard signature scheme by a factor of 85. Verification Time represents the average time per signature verification while processing a large batch. Time is normalized so one computation of a hash digest of a short message is 1 unit. The time required for the management of MAVE-3 tables is not considered, since it’s negligible if tables are stored in RAM. The ECDSA values assume that some parameters of the signature scheme are shared between different users of the system. All scheme parameters are chosen to provide security equivalent to 80 bit symmetric keys. This choice is arbitrary and we do not claim long term security using these key sizes, which serve only to allow comparison between schemes. Table 2 shows a comparison between OTP signature schemes and MAVE-3. Since MAVE-3 does not require the transmission of one time public keys, it requires a much lower signature size.

Table 1. Comparison of Standard Signature Schemes and MAVE-3

<table>
<thead>
<tr>
<th>Standard Schemes</th>
<th>PK size</th>
<th>Signature Size</th>
<th>Verification Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSA 1024</td>
<td>1024</td>
<td>320</td>
<td>1891</td>
</tr>
<tr>
<td>RSA 1024</td>
<td>3072</td>
<td>2048</td>
<td>255</td>
</tr>
<tr>
<td>ECDSA 160 over GF(p)</td>
<td>160</td>
<td>320</td>
<td>18166</td>
</tr>
<tr>
<td>MAVE-3</td>
<td>160</td>
<td>664</td>
<td>3</td>
</tr>
</tbody>
</table>

1Data is taken from Crypto++ 5.6.0 Benchmarks
Table 2. Comparison of OTP Signature Schemes and MAVE-3

<table>
<thead>
<tr>
<th>OTP schemes</th>
<th>Signature size (OT PK+OTS)</th>
<th>Verification Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lamport</td>
<td>19200</td>
<td>10</td>
</tr>
<tr>
<td>Merkle-Winternitz</td>
<td>1920</td>
<td>169</td>
</tr>
<tr>
<td>Bleichenbacher-Maurer</td>
<td>3680</td>
<td>72</td>
</tr>
<tr>
<td>BiBa 1024</td>
<td>82800</td>
<td>23</td>
</tr>
<tr>
<td>PowerBall 1024</td>
<td>82720</td>
<td>20</td>
</tr>
<tr>
<td>MAVE-3</td>
<td>664</td>
<td>3</td>
</tr>
</tbody>
</table>

2. Digital Signature Protocols and Third Parties

The problem of achieving short and fast digital signatures can be solved by switching from a digital signature algorithm to a digital signature protocol and from asymmetric algorithms to hash based (one-time) schemes, but one that does not require a one-time public key. In this paper we propose new digital signature protocols (the MAVE family) that conforms to these guidelines and can provide massive verification capability without requiring a permanent trusted third party. MAVE also requires third parties (called Aggregators) but puts no special trust condition for them. The only condition is that the aggregators are uniquely identifiable by a single message that they broadcast and are responsible and accountable for the messages they sign. By identifiable we mean that every user in the network agrees that the message is authentic by looking at the message received without previous interaction. The aggregator role can be fixed, it can be chosen randomly over the existing users at different times, or it can be assigned dynamically to a user that achieves greater score or certain threshold for any defined metric. The Bitcoin protocol [17] uses a proof-of-work as a metric to elect a signature aggregator, which is called Miner. MAVE does not specify how the network must deal with competition among users to become the aggregator nor the protocol to achieve consensus over who the aggregator is.

The aggregator in MAVE cannot impersonate a user without alerting every other user in the network of the malicious activity. MAVE specifies that the first document signed with a given one-time key is the valid document and any remaining use of the key implies forgery. To achieve consensus on which of two commands appeared first, each dated block specifies the hash of a previous block, building a chain, as in a linked time-stamping service.

MAVE-2 works best when the aggregator is fixed by the system or changes rarely. In such cases, using symmetric authentication using MAC keys between normal users and the aggregator is possible. MAVE-3 on the contrary, works best when there is no previous relation between the aggregators and the remaining users, and when such a previous relation is discouraged. For example, when the network tries to avoid a single point of failure to reduce the chances of a DoS attack. At any time, one or more users may be competing to acquire the aggregator role. Since the later scenario is more demanding in terms of security, we’ll present MAVE-3 in detail and only briefly MAVE-2. Both protocols will be referred as MAVE when specifying common properties.

Preliminary version – April 17, 2012
3. Blocks, Commands and Signatures

The aggregator role consists of publishing one or more blocks. Every user in the network must be able to receive the published blocks, either through a broadcast medium, a store and forward network or a service that allows users to anonymously access a public log file. We require that no receiver can be isolated or skipped, so every user is able to receive every block in a timely manner with high probability. A block contains a list of commands that have been sent by the users to the aggregator (or to the candidates to be an aggregator before the role has been assigned). The commands can be sent using the same medium used for block publishing or some other medium. Each block must be able to be recognized by the network as authentic (coming from an aggregator). In MAVE-2, the aggregator is publicly known in advance, so the block can be signed by the aggregator with a classical asymmetric digital signature algorithm. Also, in MAVE-2, users are required to mutually authenticate with the aggregator each time they send a command, but not necessarily with classical digital signatures. The authentication can be carried on with private key algorithms such as using a keyed MAC. The aggregator in MAVE-2 therefore requires slightly higher resources than the remaining users, to carry on authentication procedures, but is not required to serve the users using encrypted or signature authenticated communication channels (such as SSL). In MAVE-3 all communications between users and the aggregators can be sent in clear-text, without authentication, without degrading the protocol security. For this reason the MAVE-3 aggregators do not require higher resources than an average user.

Each MAVE block published by an aggregator contains user commands. A MAVE signature consists of 2 or 3 commands (MAVE-2 and MAVE-3 respectively). Each command is published in a different block. The user is responsible for sending these commands, one by one, to the known aggregator or to the candidates to be the next aggregator. Only after the third command is published in a block, the signature is complete.

When a command is included in a block, and the block is broadcast, we say the command has been issued. Only the last command of a signature actually issues the signature. Before the last command, a document is not considered signed. The previous commands only serve to associate a (still unpublished) key to a particular signature so when the last command containing the key is issued there is no doubt which is the right signature to accept. Counterfeit commands, although they may contain a valid key, will not be accepted since a genuine and previously issued association will exist.

Although MAVE-2 requires less commands per signature (2 instead of 3), MAVE-2 is not suited for situations where the aggregator is not previously known. If MAVE-2 is used without user-to-aggregator authentication, then the system will suffer from the $O(c^2)$ attack (see section 11.3) or a delay attack (see section 11.1) depending on variations of the checking procedures used.

If MAVE protocols are implemented in a way that there is competition for the aggregator role, then it’s important that the signers don’t send their last command until any block chain conflict is resolved in favor of the block chain with highest score and that score is much higher than the score of the competing chains. If users only check that the commands have been issued in any alternate block chain, they would allow the reuse of the keys to sign other documents in the main block chain.
4. Key distribution

Every MAVE user knows the public keys of remaining users. The key distribution problem should be solved by other means, such as a trusted key directory or any other PK infrastructure. It also can be solved by delegation of trust using the protocol itself. A known user (whose public key has been verified) can sign a message publishing the public key and the ID of a user unknown to the receiver.

5. Additional Requirements

One requirement for the digital signature protocol to be practical is that signatures can be interleaved. In MAVE terms, it should be possible to start a new signature while not all the commands of a previous signature have been sent to the aggregator. Because MAVE signatures span multiple blocks, we must explicitly design for interleaving. This is achieved in MAVE by updating the next available key as soon as possible, and keeping records of active unfinished signatures.

Last, to boost signature verification performance, MAVE can use different key sizes for different public keys. If the network can relate an increasing signature size with a higher cost, then using different key sizes will reduce the average bandwidth required, giving each user a security threshold according to the real security requirements of the signed documents involved. Also to improve performance, temporary and permanent data used by client applications can be stored in RAM and flushed periodically to disk.

6. User key chain

Each user has at least one public key, which is build using a key chain. To generate a key chain we need a one-way function (computationally unfeasible to invert). For this purpose we can use standard cryptographic hash function. Each standard hash function has a specified signature length (N). Some common hash functions are SHA-1 (N=20), SHA-256 (N=32) and RIPEMD-128 (N=16). The first key is a preimage of the public key, and each the following key is a preimage of the previous key. As we do not require the collision resistance property of the hash function, a hash function with N>=10 will suffice for signatures that represent low monetary value. For signatures of higher value, we can choose N>=20.

If we want to provide keys of short length (N<16) then no secure hash function will be found, since hash functions require collision resistance. We can still achieve shorter sizes than the hash digest of a widely analyzed hash function by truncating the digest down to the desired bits. For example, if 80-bit security is desired, the first key would be a 80-bit value that, when padded with zeros, is the preimage of the digest whose 80-bit prefix is the public key. Consequently, the following key would be a 80-bit value that, when padded with zeros, is the preimage of a digest whose 80-bit prefix is the current key.

A one-way function is also required for deterministic commitments. If a hash function is used, we’ll show how we can use truncated hash digests to achieve a reduction in the size of the commitments.

---

2The hash digest truncation is insecure if quantum computing becomes practical, since according to Grover’s algorithm, finding a preimage collision on a single invocation of an ideal hash function is bounded on O(2^n/2) operations under a quantum computing model.
MAVE-3 is a digital signature protocol that allows massive verifications, requires a third party but does not impose the users to know of who the aggregator might be.

The key MAVE-3 properties are:
1. Unauthenticated communication free of risk
2. Interleaved signatures
3. Protection against the Delay Attack (see section 11.1)
4. Protection against the $O(c^2)$ Attack (see section 11.3)
5. Protection against Targeted DoS Attack (see section 11.2)
6. Arbitrary length payload
7. No previous knowledge of aggregator identity is required

The fact that MAVE-3 signatures can carry an arbitrary length payload allows MAVE to be used to transmit authenticated data instead of an authenticated hash of the data. This is desirable if the payload size is small compared with the size of a hash digest and all authenticated data is required by all users in real time.

To cancel a payment before it has been completely issued, the sender simply avoids broadcasting the last command. Because MAVE have timeouts for temporary data retention, after some blocks the unfinished signatures are automatically canceled without blocking further transactions.

Figure 1 shows how MAVE-3 key chains are built. Figure 2 shows how MAVE-3 commands and MAVE-3 key are related.

### 7.1. Public key Creation

A MAVE public key is built by applying multiple times a hash function to a seed message, creating a hash chain. The first message is a random or psudorandom seed. The last message is the signature public key. A preimage of the public key under the hash function is the first signing key. The preimage of the first signing key is the second, and so on. The first key (the seed) is the last signing key that can be used. As signing keys are disposable, each user must track the next signing key for an specified public key. We use a keyed Message Authentication Code (MAC) to derive a second chain from the main chain, to avoid storing additional seed messages. We suggest using HMAC [16].

**Def. Account:** For a private key $(K_1(1), N_K, Q)$, the public key is $S$, and the following relations must hold:
- $N_K$ is the number of keys in the chain.
- $K_1(1)$ is the first key for the chain $K_1$, a random or pseudo-random N-byte string.
- $K_2(i) = NextKey(HMAC_Q(i))$ for $1 \leq i \leq N_K$
- $K_P(i) = NextKey(K_2(i))$ for $1 \leq i \leq N_K$
- $K_1(i) = NextKey(K_1(i-1)||K_P(i-1))$ for $2 \leq i \leq N_K$
- $S = NextKey(K_1(N_K)||K_P(N_K))$.

### 7.2. Signing protocol

$k_1$: the next key of the chain $K_1$ that will be accepted, related to the public key $S$. 

Preliminary version – April 17, 2012
$k_2$: the next key of chain $K_2$, related to $k_1$, that can be dynamically computed from the key index number.

We define a commitment scheme ($Commit$) based on a hash function. The scheme must not be equal to the account number generation scheme and must be deterministic. It’s also possible to truncate the hash digest to achieve shorter commitments, since collision resistance is not required. The commitment scheme is:

**Def. Commit**: The function $Commit$ is defined as follows:
- $Commit(msg) = \text{Truncate}(\text{Hash}(\text{COMMIT}||\text{Hash}(msg)), N)$.
- $N$ is the desired length of the commitment in bytes.

The block format is:
- $\text{block}(i) = < \text{block\_data, pk\_block\_signature} >$
- $\text{block\_data} = < \text{command\_list, prev\_block\_hash}, i >$
- $\text{prev\_block\_hash} = \text{Hash}(\text{block}(i-1))$
- $N_C$ is number of commands in the block number $i$.
- $\text{command\_list} = < \text{command}(1), ..., \text{command}(N_C) >$

Preliminary version – April 17, 2012
To sign a document $X$ the signor must privately build the 3 commands ($m_1$, $m_2$ and $m_3$), each one of a distinct type. To identify command types we use the identifiers $TYPE1$, $TYPE2$ and $TYPE3$. These identifiers can be encoded as a single byte field. We show the format of each command, in the order of computation of the fields.

$$m_3 = \langle TYPE3, k_2 \rangle$$
$$payload = \langle H(X) \rangle$$
$$cm_3 = Commit(m_3)$$
$$m_2 = \langle TYPE2, S, payload, k_1, k_P, cm_3 \rangle$$
$$cm_2 = Commit(m_2)$$
$$m_1 = \langle TYPE1, cm_2 \rangle$$

Commands are build in the reverse order they are sent to the aggregator. To make the signature public, the signor must:

1. Compute $m_3$, $m_2$ and $m_1$.
2. Broadcasts $m_1$ to the network.
3. Waits for $m_1$ to be included in a block issued by a valid aggregator.
4. Broadcasts $m_2$.
5. Waits for $m_2$ to be included in a block issued by a valid aggregator.
7. Waits for $m_3$ to be included in a block issued by a valid aggregator.

At this point, the signature is considered complete.
7.3. Signature verification protocol. To achieve massive bulk signature verifications, it’s crucial that the process of verifying a signature can be carried out efficiently. Processing a signature means linking commands to signatures, linking signatures to public keys, and linking public keys to individuals. To achieve this goal, the client application must maintain 4 data structures: \textit{TABLE} \_1, \textit{TABLE} \_2, \textit{OPEN} \_\textit{SIGNATURES} and \textit{KEYRING}.

- \textit{KEYRING} has records $< I D, S, Lk_1 >$, that matches each public key $S$ with the owner identified by ID and $Lk_1$, which is the last account key of the chain $k_1$ associated with the public key. The data structure should allow efficient access by the key $S$.
- \textit{TABLE} \_1 contains records $< cm_2, bn >$, where $cm_2$ is a commitment to a command type 2 and $bn$ is the block number, indexed by $cm_2$.
- \textit{TABLE} \_2 contains records $< Sk, cm_3, m_2, bn >$, where $Sk$ is the public key $S$ concatenated with a key $k_1$, $cm_3$ is a commitment to a command of type 3, $m_2$ is a command of type 2 and $bn$ is the block number where the command was found. The table is indexed by $Sk$ and by $cm_3$.
- \textit{OPEN} \_\textit{SIGNATURES} contains records $< k_1, bn >$, where $k_1$ is a key of an account and $bn$ is the block number where the key was first found. The table is indexed by $k_1$.

\begin{table}[h]
\begin{tabular}{|l|}
\hline
\textbf{Procedure CHECK\_BLOCK} \\
\hline
When a block $\text{block}(i)$ arrives each client must execute the following procedure. \\
\hline
(1) Check the validity of the aggregator. \\
(2) Check the block date. It must occur after the date of the previous block. \\
(3) if $i > 0$ then check that $\text{prev\_block\_hash} = \text{Hash(\text{block}(i - 1))}$. If not, then abort. \\
(4) Check that there is no other block with index $i$ previously accepted. If so, resolve the conflict between competing aggregators. \\
(5) For each command in the block, in order of appearance, and depending on the command type, call CHECK\_TYPE1, CHECK\_TYPE2 or CHECK\_TYPE3. \\
(6) If none of the command checking procedures has aborted, accept the block as valid. \\
\hline
\textbf{Procedure CHECK\_TYPE1} \\
Let $m_1 = < \text{TYPE1}, cm_2 >$ be a command $m_1$ of type 1 that is included in the block number $bn$. \\
(1) Save the the record $< cm_2, bn >$ in a table \textit{TABLE} \_1 (efficiently indexed by the field $cm_2$). \\
\hline
\end{tabular}
\end{table}
Subprocedure CHECK_FORGED_SIGNATURE($m_2, cm_2, bn_1, bn_2$)

(1) Check if a record with key $Sk = S|k_1$ exists in $TABLE_2$.
(2) If a record exists:
   (a) Let the record found in a table $TABLE_2$ be $< Sk, cm'_3, m'_2, bn'_2 >$
   (b) Compute $cm'_2 = Commit(m'_2)$
   (c) Check if a record with key $cm'_2$ is found in $TABLE_1$.
   (d) If not, then the tables are corrupted. Abort.
   (e) Let the record found $TABLE_1$ be $< cm'_2, bn'_1 >$
   (f) If $bn'_1 < bn_1$ then abort processing since the command is counterfeit.
   (g) Remove the record with key $Sk$ from $TABLE_2$.
   (h) Remove the record with key $cm'_2$ from $TABLE_1$.
(3) Insert the record $< Sk, cm_3, m_2, bn_2 >$ in $TABLE_2$.

Subprocedure CLEAN_UP($m_2, m_3$)

(1) Remove the records in table $OPEN_SIGNATURES$ with key $k_1$. This will prevent any forged transaction to try to use the key $k_1$ again.
(2) Remove the two entries from $TABLE_1$ and $TABLE_2$ that were found related to $m_3$.

Subprocedure SET_NEXT_KEY($S, k_1, Nk_1$)

(1) Search the table $KEYRING$ using $S$ as indexing key. If no record is found and all peers are known in advance then the command is invalid. If we accept signatures of unknown users, and no record is found, then just skip this command.
(2) Let $X = < S, id, Lk_1 >$ be the record found on the previous step.
(3) if $Lk_1 = Nk_1$ then
   (a) Replace record $X$ with record $< S, id, k_1 >$ in $KEYRING$.
   (b) Insert the record $< k_1, bn >$ in table $OPEN_SIGNATURES$.
(4) else
   (a) Search the table $OPEN_SIGNATURES$ for the record with key $k_1$.
   (b) If the record is not found then abort.

Procedure CHECK_TYPE2

Let $m_2 = < TYPE2, S, H(X), k_1, k_p, cm_3 >$ be a command of type 2 that is included in the block number $bn$.

(1) Compute $cm_2 = Commit(m_2)$
(2) Check if a record with key $cm_2$ is found in $TABLE_1$. If not, then the command is invalid, and so the block containing it.
(3) Let the record found $TABLE_1$ be $< cm_2, bn_1 >$.
(4) Set $Nk_1 = NextKey(k_1||k_p)$
(5) Call SET_NEXT_KEY($S, k_1, Nk_1$)
(6) Call CHECK_FORGED_SIGNATURE($m_2, cm_2, bn_1, bn$)
Procedure CHECK_TYPE3

Let $m_3 = <TYPE3, k_2>$ be a command of type 3 that is included in the block number $bn$.

(1) Let $cm_3 = Commit(m_3)$. Let $x_2$ be the record in $TABLE_2$ found with key $cm_3$ (using the appropriate index). If no record is found, then abort.
(2) Let $x_2 = <Sk, cm_3, m_2, bn_2>$.
(3) Let $m_2 = <TYPE2, S, H(X), k_1, k_P, cm_3>$.
(4) If $k_P \neq NextKey(k_2)$ then abort.
(5) Search the table $KEYRING$ using $S$ as indexing key.
(6) If no record is found, skip up to the clean up step.
(7) Let $X = <S, id, Lk_1>$ be the record found. Record the fact that document $H(X)$ has been signed by $ID$ at the date of block $bn$.
(8) Call CLEAN_UP($m_2, m_3$)

The client application must also periodically search $TABLE_1$, $TABLE_2$ and $OPEN_SIGNATURES$ for entries that are to old to be allowed to finish. For example, the protocol could establish a limit of 100 blocks for the lifetime of any unfinished signature.

The MAVE-3 protocol allows interleaved signatures, up to a maximum of one signature per block. This limit can be easily overcome if a key index is included in the type 2 command and the last used key index is included in the $KEYRING$ record. In this case, the network must specify a maximum number of interleaved transactions that will be supported per block, to avoid DoS attacks by forcing excessive processing by the clients when checking for validity of a given key in the chain.

It is possible to design a protocol similar to MAVE-3 that does not require a one-way hash function with second preimage resistance, but does not allow overlapping nor cancellable signatures.

7.4. Examples. Here is an example of a signature verification of a MAVE-3 signature without a forgery attempt.

(1) The initial state of client tables is:

<table>
<thead>
<tr>
<th>Table</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$KEYRING$</td>
<td>$&lt;S', JohnSmith', k_1&gt;$</td>
</tr>
<tr>
<td>$TABLE_1$</td>
<td>empty</td>
</tr>
<tr>
<td>$TABLE_2$</td>
<td>empty</td>
</tr>
<tr>
<td>$OPEN_SIGNATURES$</td>
<td>empty</td>
</tr>
</tbody>
</table>

(2) Block 1 arrives containing the command $m_1 = <TYPE1, Commit(m_2)>$

(3) After processing the command, the state of internal tables is:

<table>
<thead>
<tr>
<th>Table</th>
<th>Updated State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$KEYRING$</td>
<td>$&lt;S', JohnSmith', Lk_1&gt;$</td>
</tr>
<tr>
<td>$TABLE_1$</td>
<td>$&lt;Commit(m_2), 1&gt;$</td>
</tr>
<tr>
<td>$TABLE_2$</td>
<td>empty</td>
</tr>
<tr>
<td>$OPEN_SIGNATURES$</td>
<td>empty</td>
</tr>
</tbody>
</table>

(4) Block 2 arrives containing the command

$m_2 = <TYPE2, S, H(X), k_1, k_P, Commit(m_3)>$

(5) After processing the command, the state of internal tables is:
(6) Block 2 arrives containing the command \( m_3 = \langle \text{TYPE3}, k_2 \rangle \)
(7) After processing the command, the state of internal tables is the same as the initial state:

<table>
<thead>
<tr>
<th>KEYRING</th>
<th>(&lt; S', JohnSmith', k_1 &gt;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_1</td>
<td>(&lt; \text{Commit}(m_2), 1 &gt;)</td>
</tr>
<tr>
<td>TABLE_2</td>
<td>(&lt; S[k_1, \text{Commit}(m_3), m_2, 2] &gt;)</td>
</tr>
<tr>
<td>OPEN_SIGNATURES</td>
<td>(&lt; k_1, 2 &gt;)</td>
</tr>
</tbody>
</table>

8. MAVE-2

MAVE-2 is a digital signature protocol that allows massive verifications and requires a third party with a previous established trust relation with users.

The key MAVE-3 properties are:

1. Authenticated communication between users and aggregators required
2. Interleaved signatures
3. Protection against the Delay Attack
4. Protection against the \(O(c^2)\) Attack
5. Arbitrary length payload

8.1. Public key Creation. A MAVE public key is built similar to MAVE-3.

8.2. Signing protocol. \(k_1, k_2\) and the block format are defined similar to MAVE-3. To sign a document \(X\) the signor must privately build the 2 commands \((m_{12} \text{ and } m_3)\), each one of a distinct type. To identify command types we use the identifiers \(\text{TYPE12}, \text{TYPE3}\). These identifiers can be encoded as a single byte field. We show the format of each command, in the order of computation of the fields.

\[
m_3 = \langle \text{TYPE3}, k_2 \rangle \\
payload = \langle H(X) \rangle \\
cm_3 = \text{Commit}(m_3) \\
m_{12} = \langle \text{TYPE2}, S, \text{payload}, k_1, k_P, cm_3 \rangle
\]

Commands are built in the reverse order they are sent to the aggregator. To make the signature public, the signor must:

1. Compute \(m_3, m_{12}\).
2. Broadcasts \(m_{12}\) to the network.
3. Waits for \(m_{12}\) to be included in a block issued by a valid aggregator.
4. Broadcasts \(m_3\).
5. Waits for \(m_3\) to be included in a block issued by a valid aggregator.

At this point, the signature is considered complete.

8.3. Signature verification protocol. The client application must maintain 3 data structures: \(\text{TABLE\_12}, \text{OPEN\_SIGNATURES}\) and \(\text{KEYRING}\).
• **KEYRING** has records $<ID, S, Lk_1>$, that matches each public key $S$ with the owner identified by $ID$ and $Lk_1$, which is the last account key of the chain $k_1$ associated with the public key. The data structure should allow efficient access by the key $S$.

• **TABLE_12** contains records $<cm_3, m_{12}, Sk, bn>$, where $bn$ is the block number where the command was found, $cm_3$ is a commitment to a command of type 3, $m_{12}$ is a command of type 12, and $Sk$ is the concatenation of the users public key $S$ and the key $k_1$. The table is indexed by $Sk$ and also by $cm_3$.

• **OPEN_SIGNATURES** contains records $<k_1, bn>$, where $k_1$ is a key of an account and $bn$ is the block number where the key was first found. The table is indexed by $k_1$.

---

**Procedure CHECK_BLOCK**

When a block $block(i)$ arrives each client must execute the following procedure.

1. Check the validity of the aggregator.
2. Check the block date. It must occur after the date of the previous block.
3. If $i > 0$ then check that $prev\_block\_hash = Hash(block(i - 1))$. If not, then abort.
4. Check that there is no other block with index $i$ previously accepted. If so, resolve the conflict between competing aggregators.
5. For each command in the block, in order of appearance, and depending on the command type, call CHECK_TYPE12, or CHECK_TYPE3.
6. If none of the command checking procedures has aborted, accept the block as valid.

**Subprocedure CLEAN_UP(x12)**

1. Remove all records in table **OPEN_SIGNATURES** with key $k_1$. This will prevent any forged transaction to try to use the key $k_1$ again.
2. Remove the record $x2$ from **TABLE_12**.

**Procedure CHECK_TYPE12**

Let $m_{12} = <TYPE12, S, H(X), k_1, k_P, cm_3>$ be a command of type 12 that is included in the block number $bn$.

1. Set $Nk_1 = NextKey(k_1 || k_P)$
2. Check if a record with key field $Sk$ equal to $S||k_1$ exists in **TABLE_12**. If a record exists then abort.
3. Call **SET\_NEXT\_KEY(S, k_1, Nk_1)** (this procedure is described in MAVE-3 section)
4. Save the record $<cm_3, m_{12}, S||k_1, bn>$ in table **TABLE_12**.
Procedure CHECK_TYPE3

Let \( m_3 = \langle TYPE3, k_2 \rangle \) be a command of type 3 that is included in the block number \( bn \).

1. Let \( cm_3 = \text{Commit}(m_3) \). Let \( x_2 \) be the record in \( TABLE_12 \) found with key \( cm_3 \). If no record is found, then abort.
2. Let \( x_{12} = \langle cm_3, m_{12}, Sk, bn \rangle \).
3. Let \( m_2 = \langle TYPE2, S, H(X), k_1, k_P, cm_3 \rangle \).
4. If \( k_P \neq \text{NextKey}(k_2) \) then abort.
5. Search the table \( KEYRING \) using \( S \) as indexing key.
6. If no record is found, skip up to the clean up step.
7. Let \( X = \langle S, id, Lk_1 \rangle \) be the record found. Record the fact that document \( H(X) \) has been signed by \( ID \) at the date of block \( bn \).
8. Call \( \text{CLEAN_UP}(x_{12}) \)

Tables \( TABLE_12 \) and \( OPEN\_SIGNATURES \) must be periodically inspected by the client application. Entries that are too old to be allowed to finish must be removed.

9. MAVE-23, an hybrid protocol

MAVE-2 and MAVE-3 can coexist. Suppose that when an aggregator is elected, a number of future block slots are reserved to the aggregator. In each of the slots, the aggregator is able to publish the commands it desires. These slots are interleaved with normal (unreserved) slots. Normal slots carry only MAVE-3 commands and reserved slots can carry both MAVE-2 and MAVE-3 commands. There is still no single point of failure, since an attacker cannot predict who will be the aggregator for the unreserved slots. He can only attack the aggregators who have reserved slots and try to prevent the blocks to be issued. With this scheme, an aggregator has the option to “sell” in advance available sub-slots for commands in the reserved slots and predictably comply with users contracts. MAVE-2 commands can be included in reserved blocks, since the user already knows who the aggregator is and, if a forged type 12 command is issued, the user knows for sure that the cheater is the aggregator. Still, a forged type 12 command cannot be used to sign documents, since the user will not send the type 3 command associated with it.

10. Applications

MAVE can be applied to different real world problems. This is a list of some possible applications:

1. Massive or ultrafast auctions
2. Massive or ultrafast stock exchange
3. Massive virtual goods exchange and tracing of good owners
4. Massive payments in virtual currencies and account balance tracing
5. Massive status broadcasting for an autonomous swarm of micro-drones (or robo-bees) in a hostile environment with a high capacity wireless local area network and in the presence of active attackers.
11. Attacks

Apart from the forgery attack, there are special attacks that target specific features of MAVE-like schemes. We present the more relevant ones.

11.1. Delay Attack. A delay attack works by doing a man-in-the-middle attack between a user and the aggregator. Since in MAVE-3 communications between users and the aggregator are unauthenticated, this attack seems at first practical. The attack works as follows: the attacker, who sits between a user and the aggregator, delays sending user commands and sends counterfeit commands with the aim that they will be included in a block before the genuine ones. For a delay attack to be effective, it may need to delay one or more of the signature commands. The aim of the attacker is to sign a different document with the victim’s private key. It’s also possible for an attacker to combine commands from different signatures. This attack is prevented by design in MAVE-3 and in MAVE-2 with peer authentication.

11.2. Targeted DoS Attack. In a Targeted DoS Attack, the attacker, after listening broadcast medium of a command coming from a certain target user, quickly sends forged commands to the aggregator, in order to disrupt the processing of the genuine command. This attack works if the protocol is designed to stop going forward with the signature if a counterfeit command is found. Also the attack is effective if the target user is discouraged to continue with the signature protocol. In MAVE-3, even if users receive forged commands on the broadcast medium, they can continue with the protocol without risk.

11.3. $O(c^2)$ Attack. An $O(c^2)$ attack tries to starve the aggregator by forcing the aggregator to do a lot of work to handle very few commands. Signature commands are of two kinds: association commands and finalization commands. All but the last command are association commands, they are meant to associate a public key and some private keys with a certain document. The last command is always a finalization command. Since association commands are sent to the aggregator before the keys are actually published, an attacker can send other counterfeit association commands that associate the same public key with different documents. Later, when the keys are published in the finalization command, the users have to collect all possible associations, choose the right one (the older valid command), and discard the remaining. This processing must be redone for each invalid finalization command. Therefore, an attacker could mount a DoS attack by sending to the aggregator $O(C)$ commands which would require $O(C^2)$ verification time for the aggregator, even if all commands must be discarded as counterfeit. This attack is prevented by design in MAVE-3 and in MAVE-2 with peer authentication.

11.4. Malicious aggregator Attack. In MAVE the aggregators are forced to be benevolent because any deviation, by either publishing an invalid block or two incompatible blocks, is immediately detected by the users in the network. Nevertheless, a malicious aggregator may try these attacks:

- Prevent a signature from being validated by avoiding issuing commands from a certain user. The aggregator can claim as an alibi that he simply did not receive the commands. To mitigate this attack, several palliatives are possible. The aggregator role can be changed periodically. Users can use pseudonymous instead of real identities to avoid being identified. Last,
users could communicate with the current aggregator directly to ensure commands are properly included in following blocks.

(2) A DoS attack on the whole network by sending a valid block filled with unnecessary commands that require too much bandwidth, computing power or storage to be widely and timely accepted by the network. To prevent the attack, we can limit the resources that a block can request from each user, such as the number of commands per block. Therefore this attack can be prevented in MAVE.

12. MAVE-3 Security Proof

We prove the security of MAVE-3 against these attacks: Forgery attempt, Delay Attack, Targeted DoS Attack and $O(c^2)$ Attack. We proof the security constructively, trying the possible disruptions at every possible stage. The proof serves as an example of a transcript of a signature verification with a forgery attempt.

Suppose Alice wants to sign a document $X$ and Mallory tries to forge the signature.

(1) The initial state of client tables is:

<table>
<thead>
<tr>
<th>KEY RING</th>
<th>$&lt; S',JohnSmith',Lk_1&gt;$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_1</td>
<td>empty</td>
</tr>
<tr>
<td>TABLE_2</td>
<td>empty</td>
</tr>
<tr>
<td>OPEN_SIGNATURES</td>
<td>empty</td>
</tr>
</tbody>
</table>

(2) Alice sends the command $m1 = < TYPE1, Commit(m2) >$ to the aggregator.

(3) Mallory cannot do anything related to $m1$ that can increase his chances of forgery.

(4) Block 1 arrives containing the command $m1$

(5) After processing the command, the state of internal tables is:

<table>
<thead>
<tr>
<th>KEY RING</th>
<th>$&lt; S',JohnSmith',Lk_1&gt;$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_1</td>
<td>$&lt; Commit(m2),1 &gt;$</td>
</tr>
<tr>
<td>TABLE_2</td>
<td>empty</td>
</tr>
<tr>
<td>OPEN_SIGNATURES</td>
<td>empty</td>
</tr>
</tbody>
</table>

(6) Alice builds the command $m2 = < TYPE2,S,H(X),k_1,k_P,Commit(m3) >$ and sends it to the aggregator.

(7) Mallory intercepts the communication and prevent $m2$ to arrive to destination.

(8) Mallory builds the commands $m1' = < TYPE1,Commit(m2') >$ and $m2' = < TYPE2,S,H(X'),k_1,k_P,Commit(m3) >$.

Mallory gains nothing by changing the fields $S$, $k_1$ or $k_P$, since any modification results in the command being rejected. Also Mallory gains nothing by changing the $Cm3$ field, since she’s not able to generate a $TYPE3$ command containing a preimage of $k_P$.

(9) Mallory sends the command $m1'$ to the aggregator.

(10) Block 2 arrives containing the command $m1'$.

(11) After processing the command, the state of internal tables is:
Table 1: Commit(m2), 1 >, < Commit(m2'), 2 >

Table 2: Empty

Open_Signatures: Empty

(12) Mallory sends the command \(m_2'\) to the aggregator.
(13) Block 3 arrives containing the command \(m_2'\).
(14) After processing the command, the state of internal tables is:

<table>
<thead>
<tr>
<th>Key Ring</th>
<th>&lt; S', JohnSmith', k_1 &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>&lt; Commit(m2), 1 &gt;, &lt; Commit(m2'), 2 &gt;</td>
</tr>
<tr>
<td>Table 2</td>
<td>Empty</td>
</tr>
<tr>
<td>Open_Signatures</td>
<td>&lt; k_1, 3 &gt;</td>
</tr>
</tbody>
</table>

(15) Alice notices that a forgery attempt, but she goes on, and tries again to send \(m_2\) to the aggregator by other means.
(16) Block 4 arrives containing the command \(m_2\).
(17) After processing the command, the state of internal tables is:

<table>
<thead>
<tr>
<th>Key Ring</th>
<th>&lt; S', JohnSmith', k_1 &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>&lt; Commit(m2), 1 &gt;</td>
</tr>
<tr>
<td>Table 2</td>
<td>&lt; S[k_1, Commit(m3), m2', 3 &gt;</td>
</tr>
<tr>
<td>Open_Signatures</td>
<td>&lt; k_1, 3 &gt;</td>
</tr>
</tbody>
</table>

(18) Now tables look normal and show no trace of the forged transaction, so the forgery attack is impossible. The cost of disposing the forgery was only two table lookups, so the \(O(c^2)\) attack was not possible. The delay attack was also prevented, since the attacker could not benefit from it. The following commands are processed as normal.
(19) Alice sends \(m_3\) to the aggregator.
(20) Block 5 arrives containing the command \(m_3 = < TYPE3, k_2 >\).
(21) After processing the command, the state of internal tables is the same as the initial state.

13. Conclusion

We have presented MAVE-2, MAVE-3 and MAVE-23, Digital Signature Protocols that allow millions of users to validate thousands of signatures per second in realtime with an average desktop computer, as long as all keyring information and user related storage can be kept in RAM while the client application is online.

References


3The time it takes for the lookups still depends on the number of active unfinished signatures.


E-mail address: sergiolerner@certimix.com; sergiolerner@pentatek.com

Preliminary version – April 17, 2012