# Progressive Verification: The Case of Message Authentication
## (Extended Abstract)

Marc Fischlin[⋆]

Department of Computer Science & Engineering,
University of California, San Diego, USA

mfischlin@cs.ucsd.edu
http://www-cse.ucsd.edu/~mfischlin/

**Abstract.** We introduce the concept of progressive verification for cryptographic primitives like message authentication codes, signatures and identification. This principle overcomes the traditional property that the verifier remains oblivious about the validity of the verified instance until the full verification procedure is completed. Progressive verification basically says that the more work the verifier invests, the better can the verifier approximate the decision.

In this work we focus on message authentication. We present a comprehensive formal framework and describe several constructions of such message authentication codes, called pv-MACs (for progressively verifiable MACs). We briefly discuss implications to other areas like signatures and identification but leave it as an open problem to find satisfactory solutions for these primitives.

## 1 Introduction

Cryptographic primitives like signatures, message authentication codes or identification involve verification procedures that assure the verifier of the validity of the input. This means that the verifier performs a certain number of verification steps and finally outputs a reliable decision; the error probability of this decision is usually negligible.

Consider the following experiment. We start the verification algorithm on some instance. After some $t$ steps, e.g., after half of the full verification, we stop the algorithm and ask for a decision about the correctness of the given instance. In this case, most verification procedures cannot predict the result better than before the start when $t = 0$. We call this all-or-nothing verification: in order to give a reliable decision one must either run the full verification procedure or need not start at all. The situation is given in the left part of Figure 1.

[⋆] This work was supported by the Emmy Noether Programme Fi 940/1-1 of the German Research Foundation (DFG).

The idea of progressive verification, as displayed in the right part of Figure 1, is to relate the error probability of the decision to the running time of the verifier. Namely, progressive verification ensures that the confidence grows with the work the verifier invests. Put differently,

> The error probability of the verifier's decision decreases nontrivially with the number of performed steps of the verification procedure.

Note that *the aim of progressive verification in general is to save time for both valid and invalid inputs*. Specifically, the verifier can choose a confidence level at the outset, prematurely terminate the verification according to this level, and can finally decide about validity of the given input. Also, *the concept* is not limited to message authentication (on which we focus here) but rather *applies to verification procedures in general*.

Assuming that the verifier never rejects correct inputs we can reconceive progressive verification as a method to spot flawed inputs prematurely. Specifically, to turn such an error detection procedure into a progressive verification algorithm let the verifier, if asked at some point during the verification, simply predict authenticity of the input if no error has been found yet. Viewed this way the paradigm then says: the likeliness of rejecting fallacious inputs grows nontrivially with the performed work. Indeed we will usually adopt this more convenient viewpoint.

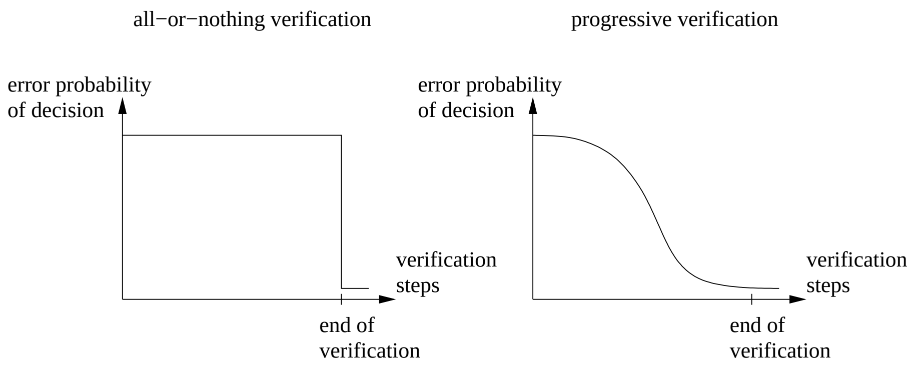

**Fig. 1.** Idea of Progressive Verification

In this work we introduce the concept of progressive verification by virtue of message authentication codes. Nowadays, all popular constructions of MACs first divide the message into blocks, then apply a pseudorandom or universal hash function to each block (possibly inserting the result from one evaluation into the input of another), and finally combine the result to a short MAC. Examples include XOR MAC [3], UMAC [5], XCBC MAC [6] and PMAC [7].

In case of such block-oriented MACs, verification progress can refer to the number of message blocks that have been inspected and for which the block function has been applied. Progressive verification then says that it suffices to look at, say, 75% of the blocks, and still be able to detect incorrect inputs quite often. Here, however, the limitations of progressive verification show. If the adversary changes only a single block then it is likely that the verifier will not be able to notice errors quite early. This is particularly true if the adversary controls the order of blocks in which the verifier receives them (for example, if the adversary delays the corresponding IP package until all other blocks have been received).

Still, the aim of progressive verification of MACs is not to spot errors with overwhelming probabilty instantaneously. It suffice to reduce the number of processed blocks for an accurate or fallacious input *on the average* —as long as the additional effort for the progressive verifiability does not outweigh this. In particular, the size of MACs should not increase too much and the extra work to compute and verify the progressively verifiable MAC should not grow significantly compared to an ordinary MAC. For example, doubling the number of block cipher invocations in order to reduce the workload for incorrect inputs by 50% is arguable.

Obviously, the overhead for progressive verification does not pay off for short messages of a few blocks only. Therefore progressive verification aims at applications where large messages are involved (like for authentication of large files); in such cases a few additional block ciphers calls merely add neglectable overhead, and doubling or tripling the short MAC of a few bits is usually not critical.

*Our Results.* We provide a formal framework for progressively verifiable MACs. This includes stringent definitions and discussions. Then we present two constructions of progressively verifiable MACs; a third one has been omitted for space reasons. One of these solutions will serve as an example to get acquainted to this field. The other one provides a reasonably good solution, allowing to spot errors after about $50\% - 55\%$ of the message while increasing the MAC size by a factor of roughly three. Note that 50% is a lower bound for such algorithms if the adversary inserts a single incorrect block only in a valid message-MAC pair: the verifier will access this block and be able to detect the error only after 50% on the average. However, if the adversary is supposed to tamper more blocks then one can go below this bound. We will touch this issue and implications to other areas like signatures at the end.

*Related Work.* Partially checking the validity of signatures or MACs has already been used in the context of incremental cryptography [1, 2]. Although the primary interest there is the fast *computation* of signatures and MACs of related messages, local checks turned out to be useful countermeasure against so-called substitution attacks. However, the results in [9, 10] indicate that checking the local validity of signatures or MACs by accessing only a few blocks yields impractically large checksums. Indeed, the idea here is similar to the incremental case: try to detect errors by inspecting only a part of the message blocks. Luckily, our aim is to detect messages as soon as possible with some (possibly small, yet) noticeable probability. By this, we can somewhat bypass the results in [9, 10] which do not give useful lower bounds for such cases.

Interestingly, some identification systems already have the property of being progressively verifiable. Namely, if the protocol proceeds in sequential rounds and in each round a cheating prover will be caught with some small probability, say, 1/2. Then there is some chance that the verifier will not have to run the whole protocol when communicating with an adversary. For instance, some identification protocols like [8] use this technique of repeating atomic protocols in order to reduce the soundness error. However, such protocols are most times superseded by identification schemes running in a single round and providing the same soundness level, like [12, 14]. Although the latter schemes are not known to support progressive verification their superior running time characteristics certainly beat the benefits of the progressive verifiability of the former schemes.

Progressive verification also raises the issue of timing attacks [13]. That is, the adversary may submit messages with fake MACs or signatures and deduce useful information from the verifier's respond time (e.g., how many blocks have been processed). Such attacks must be treated with care. In fact, we can easily extend our model such that the adversary actually learns the number of accessed blocks before rejection. Alternatively, one can try to avoid such attacks completely, i.e., by replying to the sender only after a predetermined amount of time, independent whether an error has been detected early or not.

Finally, in a related paper [11] we show how to improve the verification time for hash chains via progressive verification. In that example, the verifier determines a variable security bound at the beginning of the verification and then merely performs a corresponding fraction of the hash function evaluations, for both valid and input inputs.

*Organization.* In Section 2 we define progressive verifiable (pv) MACs. In Section 3 we present two constructions of such pv-MACs. In Section 4 we discuss extensions of our model as well as applications of the concept to other areas like signature schemes.

## 2 Definition

The definition of progressive verification mimics the one of adaptive chosen-message attacks on MAC schemes. The basic difference is how the verification proceeds and what the adversary's success goal is. Before dipping into the technical details we give a brief overview and then fill in the details.

*Overview.* For the definition we will modify the well-known adaptive chosen-message attacks scenario on message authentication codes. Let $\mathcal{A}$ be an adversary mounting such a chosen-message attack, i.e., $\mathcal{A}$ has access to oracles $\textsc{mac}$ and $\textsc{vf}$ producing, respectively, verifying MACs with some secret key. Here we substitute the oracle $\textsc{vf}$ by a so-called progressive verification step which we specify below.

Upon termination, in classical forgeability attacks the adversary's task is to come up with a new message and a valid MAC for this message. As for progressive verification, the adversary $\mathcal{A}$ now aborts the experimental phase and engages in another final progressive verification for a new message. We call this the adversary's *attempt*.

For the attempt we measure the probability that the verifier accesses at most $m$ of $n$ blocks of the adversarial message before detecting an error, i.e., after a fraction $p = \frac{m}{n}$ of all blocks. We will denote this probability by $\Delta_{\mathcal{A}}^{\mathsf{order}}(p)$. The superscript $\mathsf{order}$ indicates the order of accesses, i.e., if determined by the verifier, at random or by the adversary. Note that, in this sense, $1 - \Delta_{\mathcal{A}}^{\mathsf{order}}(1)$ denotes the probability of $\mathcal{A}$ forging a MAC for a new message.

For a progressively verifiable MAC we will demand that the detection probability is positive for some $p < 1$. This should hold for any adversary $\mathcal{A}$ bounded by a certain running time and query complexity. This means that for any such bounded adversary the verifer sometimes spots incorrect inputs before reading the whole message. Note that this definition is quite liberal: it suffices for example to identify incorrect inputs at the second to last block with some very small probability. However, the higher the probability for small fractions, the better of course.

*Progressive Verification Protocol.* We now specify the progressive verification procedure. We define it as an interactive protocol between $\mathcal{A}$ and a verifier $\mathcal{V}$ involving a third party $\mathcal{T}$. This party $\mathcal{T}$ is considered to be trustworthy, i.e., follows its prescribed program and does not cooperate maliciously with the adversary. *We note that $\mathcal{T}$ is a virtual party introduced for definitional reasons only; the actual progressive verification is of course carried out locally at the verifier's site.*

Instructively, one may think of $\mathcal{T}$ as the communication channel from the adversary's output chain to the verifier's memory. In particular, the order in which the verifier is able to process the message blocks can depend on:

– the verifier's choice, e.g., if the verifier loads the whole message into the memory and then decides on the order. Here, the verifier's choice may be fixed in advance (nonadaptive) or depend on intermediate results (adaptive). We denote these orders by $\mathsf{V.nonad}$ and $\mathsf{V.adapt}$, respectively.
– on random delays, say, depending on the transportation over the Internet. We denote this order by $\mathsf{T.rnd}$ where we presume that the distribution is clear from the context. If not stated differently, then we assume the uniform distribution on the set of all orders (although we do not claim that this is the appropriate description of delays of Internet packages).
– the adversary's decision, e.g., if the adversary delays the packages maliciously according to a nonadaptive behavior ($\mathsf{A.nonad}$). Concerning adaptive choices of the adversary we refer for the discussion following the formal description of progressive verification steps.

The fully specified progressive verification protocol is given in Figure 2. At the beginning of the procedure $\mathcal{A}$ commits to a message $M^* = m_1^*||\ldots||m_n^*$ of $n$ blocks (each block, possibly except for the final one, consisting of $B$ bits). $\mathcal{A}$ also sends a putative MAC $\tau^*$ to $\mathcal{T}$ who forwards $\tau^*$ and the length $|M^*|$ of the message to $\mathcal{V}$. Then, in each of the $n$ rounds, $\mathcal{T}$ delivers one of the message blocks to the verifier. The order of the blocks is determined accoding to the parameter order, i.e., chosen by the verifier, the adversary or at random by $\mathcal{T}$. At the end of each round $\mathcal{V}$ can return a REJECT notification to $\mathcal{T}$. Party $\mathcal{T}$ then hands the final decision to the adversary.

```
Progressive Verification for $\mathsf{order} \in \{\mathsf{V.nonad}, \mathsf{V.adapt}, \mathsf{T.rnd}, \mathsf{A.nonad}\}$

– $\mathcal{A}$ submits message $M^* = m_1^*||\ldots||m_n^*$ of $n$ blocks and putative MAC $\tau^*$ to $\mathcal{T}$
– $\mathcal{T}$ forwards $|M^*|$ and $\tau^*$ to $\mathcal{V}$
– $\mathcal{V}$ may send REJECT to $\mathcal{T}$ and stop; $\mathcal{T}$ then sends REJECT to $\mathcal{A}$

– // DETERMINE ORDER FOR NONADAPTIVE AND RANDOM ORDERS:
    if $\mathsf{order} = \mathsf{V.nonad}$ then $\mathcal{V}$ sends a permutation $\pi$ to $\mathcal{T}$
    if $\mathsf{order} = \mathsf{A.nonad}$ then $\mathcal{A}$ sends a permutation $\pi$ to $\mathcal{T}$
    if $\mathsf{order} = \mathsf{T.rnd}$   then $\mathcal{T}$ generates a random permutation $\pi$

– for $j = 1$ to $n$ do    // LET $\Pi_{j-1}^c = \{1, \ldots, n\} - \{\pi(1), \ldots, \pi(j-1)\}$

    • // DETERMINE NEXT BLOCK FOR ADAPTIVE ORDER:
        if $\mathsf{order} = \mathsf{V.adapt}$ then $\mathcal{V}$ submits $\pi(j) \in \Pi_{j-1}^c$ to $\mathcal{T}$

    • $\mathcal{T}$ forwards $(\pi(j), m_{\pi(j)}^*)$ to $\mathcal{V}$
    • $\mathcal{V}$ may send REJECT to $\mathcal{T}$ and stop; $\mathcal{T}$ then forwards REJECT to $\mathcal{A}$

– if $\mathcal{V}$ has not stopped yet then $\mathcal{V}$ sends ACCEPT to $\mathcal{T}$ who sends ACCEPT to $\mathcal{A}$
```

**Fig. 2.** Progressive Verification Protocol

There are numerous variations to the progressive verification protocol (which we do not investigate in detail in this extended abstract here). For example, one could let $\mathcal{T}$ not pass the length $|M^*|$ of the message to $\mathcal{V}$ at the outset. Also note that one could demand that $\mathcal{V}$ determines the order of accesses before even seeing the MAC $\tau^*$; indeed, in two of our solutions $\mathcal{V}$ actually bases the order on $\tau^*$. Also, to capture timing attacks, one may let $\mathcal{T}$ forward the decision together with the round number $j$ to the adversary upon rejection.

Furthermore, in each loop $\mathcal{A}$ could be informed by $\mathcal{T}$ about the block number sent to $\mathcal{V}$. In this case, one could for example let the adversary at the end of each round decide to alter message blocks that have not been delivered yet. Depending on whether the message length is given to $\mathcal{V}$ in advance one could then also let the adaptive adversary decide to change the length of the message during the rounds (but such that the length never drops below a previously delivered block number). Or, instead of letting the adversary decide maliciously where to put incorrect message blocks, those message blocks can be disturbed accidently.

*Restrictions on the Attempt.* We restrict the input that the adversary can use in her final attempt after the experimental phase. Namely, we demand that the adversary's message $M^*$ has not been submitted to MAC earlier. This is the classical restriction from unforgeability definitions of MACs. Here, this allows to measure how well the progressive verifcation works for tampered inputs.

*Detection Probabilities.* Let $\mathbb{Q}_{[0,1]} := \mathbb{Q} \cap [0,1]$. For $p \in \mathbb{Q}_{[0,1]}$ denote by $\Delta^{\mathsf{order}}_{\mathcal{A}}(p)$ the probability that $\mathcal{V}$ outputs REJECT after having accessed $m = \lfloor pn \rfloor$ blocks in the attempt with adversary $\mathcal{A}$. Obviously $\Delta^{\mathsf{order}}_{\mathcal{A}}(0) = 0$. Moreover, $\Delta^{\mathsf{order}}_{\mathcal{A}}(1)$ equals the probability that the adversary will be caught at all. We call $\Delta^{\mathsf{order}}_{\mathcal{A}}(p)$ the *cumulative detection probability* as it describes the probability of finding errors with a fraction of $p$ *or even less* blocks.

The cumulative detection probability $\Delta^{\mathsf{order}}_{\mathcal{A}}$ is defined with respect to a specific adversary $\mathcal{A}$. To extend this defintion to sets of adversaries we parameterize adversaries by a vector $(t, \boldsymbol{q}, \boldsymbol{b})$ describing the running time $t$ of the adversary (in some fixed computational model), the number of queries $\boldsymbol{q} = (q_{\text{MAC}}, q_{\mathcal{V}})$ to MAC and $\mathcal{V}$, and the maximum number of blocks $\boldsymbol{b} = (b_{\text{MAC}}, b_{\mathcal{V}})$ in each submission. Such an adversary is called $(t, \boldsymbol{q}, \boldsymbol{b})$*-bounded.*

We let $\Delta^{\mathsf{order}}_{(t,\boldsymbol{q},\boldsymbol{b})}(p)$ be a function such that for any $(t, \boldsymbol{q}, \boldsymbol{b})$-bounded adversary $\mathcal{A}$ we have

$$\Delta^{\mathsf{order}}_{(t,\boldsymbol{q},\boldsymbol{b})}(p) \leq \Delta^{\mathsf{order}}_{\mathcal{A}}(p) \qquad \text{for all } p \in \mathbb{Q}_{[0,1]}.$$

Note that with $\Delta^{\mathsf{order}}_{(t,\boldsymbol{q},\boldsymbol{b})}(p)$ the function $\Delta^{\mathsf{order}}_{(t,\boldsymbol{q},\boldsymbol{b})}(p)/2$ for example is also an appropriate function. Of course, we usually seek the best security bound. It is also straightforward to derive a classical asymptotic definition relating the adversary's parameters polynomially to a security parameter.

We remark that there are some subtle points in the definition here. For example, if we consider an adversary $\mathcal{A}$ that always outputs a message of two blocks in her attempt, then $\Delta^{\mathsf{order}}_{\mathcal{A}}(p) = 0$ for $p < 0.5$ because $\lfloor np \rfloor = 0$ for such values $n = 2$, $p < 0.5$. Therefore this bound carries over to the set of adversaries and $\Delta^{\mathsf{order}}_{(t,\boldsymbol{q},\boldsymbol{b})}(p) = 0$ for $p < 0.5$ . Thus, usually some restriction on the length of the message blocks applies, say, the progressive verification procedure is only run on messages of $n \geq 100$ blocks to provide a sufficient granularity of 1%. Since our solutions build on top of an ordinary MAC we can simply run the basic verification procedure if $n < 100$ and invoke the progressive verification for $n \geq 100$.

Formally, we can include a lower bound $b^L_{\mathcal{V}}$ of the number of blocks for progressive verification runs in the bound $(t, \boldsymbol{q}, \boldsymbol{b})$ on the adversary. Then we can "adjust" the small error caused by the granularity by subtracting $1/b^L_{\mathcal{V}}$ from $\Delta^{\mathsf{order}}_{(t,\boldsymbol{q},\boldsymbol{b})}(p)$.

*Defining Progressively Verifiable MACs.* Each "ordinary" MAC can be considered as a pv-MAC with $\Delta^{\mathsf{order}}_{(t,\boldsymbol{q},\boldsymbol{b})}(p) = 0$ for $p < 1$. We thus rule out such trivial solutions in the following definition; yet, we merely demand that premature error detection may happen sometimes:

**Definition 1.** *Let* $\mathsf{order} \in \{\mathsf{V.nonad}, \mathsf{V.adapt}, \mathsf{T.rnd}, \mathsf{A.nonad}\}$. *Then a message authentication code is* $(t, \boldsymbol{q}, \boldsymbol{b})$*-progressively verifiable with respect to* $\mathsf{order}$ *if there is a function* $\Delta^{\mathsf{order}}_{(t,\boldsymbol{q},\boldsymbol{b})}$ *with*

$$\Delta^{\mathsf{order}}_{(t,\boldsymbol{q},\boldsymbol{b})}(p) > 0$$

*for some* $p < 1$.

It is again easy to infer an asymptotic definition.

We sometimes call a MAC which is $(t, \boldsymbol{q}, \boldsymbol{b})$-progressively verifiable with respect to some order simply a pv-MAC, prescinding the bound on the running time as well as the type of order. Accordingly, we sometimes write $\Delta^{\mathsf{order}}(p)$ or even $\Delta(p)$ etc. if the details are irrelevant or clear from the context.

From the cumulative detection probability $\Delta(p)$ one can deduce the corresponding density function $\delta(p)$ describing the detection probability after reading exactly $m = pn$ of $n$ blocks. This also allows to define the average number of blocks the verifier accesses. We provide the formal definitions in the full version.

## 3 Constructions of pv-MACs

We start with a straightforward approach to warm up. We then move on to a more sophisticated approach in Section 3.2. Interestingly, our solutions below do not exclude each other: in fact, all solutions can be combined easily to provide improved detection performance. All solutions work for a nonadaptively chosen order by the verifier.

### 3.1 Divide-and-Conquer Construction

The divide-and-conquer method works as follows. Assume that we are given some secure MAC scheme. For a message of $n$ blocks (where we assume for simplicity that $n$ is always even), instead of MACing all $n$ blocks together, we individually compute a MAC for the first $n/2$ blocks and then for the remaining $n/2$ blocks where we prepend the fixed-length MAC of the first half to the message (possibly padded to block length). This is done with independent keys for each part. The complete MAC is given by the concatenation of both individual MACs.

We assume that the verifier determines the order of block accesses. That is, the verifier tosses a coin and, depending on the outcome, first starts to read all the blocks of the left or right half of the message. The verifier then checks the MAC of this part (by prepending the putative MAC of the left half if the right half is checked). If this MAC is valid then the verifier continues with the other half and verifies the other MAC. $\mathcal{V}$ accepts if and only if both tests succeed.

It is not hard to see that the verifier will detect errors after reading half of the input with probability 0.5, except if the adversary $\mathcal{A}$ manages to forge a MAC of the underlying scheme which happens with some small probability $\epsilon_{\mathcal{A}}$ only. We omit a formal proof in this version.

Altogether, the cumulative detection function $\Delta_{\mathcal{A}}^{\mathsf{V.nonad}}(p)$ for this progressive verification with nonadaptively chosen order is given by:

$$\Delta_{\mathcal{A}}^{\mathsf{V.nonad}}(p) = \begin{cases} 0 & \text{if } 0 \leq p < 0.5 \\ 0.5 - 2\epsilon_{\mathcal{A}} - \frac{1}{b_{\mathcal{V}}^{L}} & \text{if } 0.5 \leq p < 1 \\ 1 - 2\epsilon_{\mathcal{A}} - \frac{1}{b_{\mathcal{V}}^{L}} & \text{if } p = 1 \end{cases}$$

Neglecting $\epsilon_{\mathcal{A}}$ and the lower bound $\frac{1}{b_{\mathcal{V}}^{L}}$ on the block number the verifier therefore reads on the average 75% of the blocks.

There are several drawbacks nested in the divide-and-conquer approach. The main drawback is that the expected number of inspected blocks for fallacious inputs is quite large: about 75% of the input have to be processed on the average. Also, the cumulative detection function is constant in the intervals $[0, 0.5)$ and $[0.5, 1)$. This is even true if the adversary tampers more than a single block in a given message-MAC pair. Then the verifier still needs to process at least 50% of the input —even if the adversary submits a random message with a random value as MAC. It is preferrable of course to be able to detect such completely faulty inputs earlier.

There are some advantages to the divide-and-conquer method, though. First, it works with any underlying MAC scheme. Second, the probabilistic verifier decides upon the order at random for each new progressive verification run. Hence, even if the adversary, in a timing attack, gets to know the number of processed blocks in previous runs, the error in the attempt will be found with probability approximately 0.5 after half of the blocks.

Certain variations to the basic scheme apply, of course. For instance one can divide the message into three equally large parts and output three MACs allowing to spot incorrect inputs after 66.66% of the blocks and so on. However, we are still left with the problem of a mainly constant cumulative detection probability.

## 3.2 Partial-Intermediate-Result Construction

In this section we address a solution where the cumulative detection probability is roughly proportional to the work. For this we first recall some basics about CBC-MACs before describing our partial-intermediate-result construction.

*Preliminaries.* To build a MAC we essentially compute an XCBC-MAC [6] of the message, using a block cipher $E_K : \{0,1\}^B \to \{0,1\}^B$ and keys $K \in \{0,1\}^k$, $K_2, K_3 \in \{0,1\}^B$ for input/output length $B$. Specifically, the XCBC-MAC of a message $M = m_1|| \ldots ||m_n$ of $B$-bit blocks $m_1, \ldots, m_n$ (with $|m_n| \leq B$) is given by:

- if $|m_n| = B$ then let $K^* = K_2$
  else if $|m_n| < B$ then let $K^* = K_3$ and pad $M$ with $10^j$ for $j = B - 1 - |m_n|$
- set $C_0 = 0^B$ and for $j = 1$ to $n-1$ compute $C_j = E_K(C_{j-1} \oplus m_i)$

– return $C = E_K(K^* \oplus C_{n-1} \oplus m_n)$ where $m_n$ has possibly been padded

That is, one pads the message only if necessary and, depending on this padding, one inserts the key $K_2$ or $K_3$ in the final step of the CBC-MAC computations.[1]

*Outline.* Instead of starting the XCBC-computation with the first message block here we start at a random bit position start $\in \{0, 1, \ldots, |M| - 1\}$ of the (unpadded) message $M$. That is, we run the XCBC-MAC computation for the rotated message with offset start, and we also prepend start (padded to $B$ bits) to this rotated message. The value start will later be appended in encrypted form to the original MAC as well.

During the MAC computation we write down the least significant bits of the intermediate results $C_{\lfloor j \cdot \text{width} \rfloor + 1}$ for width $= \frac{n}{1+B/2}$. We call the corresponding message blocks *check blocks.*[2] By the choice of the value width we get $B/2$ extra bits lsb which can then be encrypted as a single block together with the $B/2$ bits representing the value start. We assume a lower bound of $b_V^L \geq 1 + B/2$ on the block length of progressively verified messages to ensure that all check blocks are distinct. Figure 3 shows the computation of the XCBC-MAC in our case.

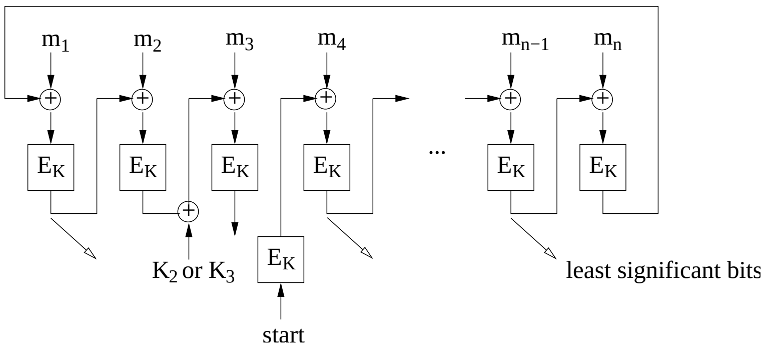

**Fig. 3.** Progressively Verifiable PIR-XCBC-MAC computation

In addition we compute another (CBC-)MAC $\sigma$ for the XCBC-MAC $C$, the encryption $e$ of lsb||start and the bit length of the message. In this case, a CBC-MAC suffices as we apply this MAC computation only to fixed length inputs of three blocks. Our complete MAC is given by $(C, e, \sigma)$.

To verify the MAC $(C, e, \sigma)$ progressively, first check the CBC-MAC $\sigma$. If it is valid then decrypt lsb||start and start re-computing the XCBC-MAC beginning

[1] We remark that any (reasonable) kind of CBC-MAC works for our construction here; we use the XCBC-MAC instead of the CBC-MAC in order to deal with variable input length right away.

[2] The reason for taking the blocks with number $\lfloor j \cdot \text{width} \rfloor + 1$ instead of $\lfloor j \cdot \text{width} \rfloor$ is that we have prepended the value start as the new first block.

at the random position start. Each time after processing a further check block compare the least significant bit of this check block with the decrypted value; if they match then continue, else stop with output REJECT. The formal description of our construction PIR-XCBC-MAC (for *p*artial *i*ntermediate *r*esults) appears in Figure 4.

```
PIR-XCBC-MAC for V.nonad order

– Key Generation: pick $k$-bit keys $K$ and $B$-bit keys $K_2, K_3, K_{\text{ENC}}, K_{\text{CBC}}$

– Compute MAC for message $M$ of $n$ blocks:
    • pick random value $\mathsf{start} \in \{0, 1, \ldots, |M|-1\}$, encode with $B/2$ bits
    • rotate $M$ by offset $\mathsf{start}$ and prepend block $0^{B/2}||\mathsf{start}$ to get $M_{\mathsf{start}}$ of $n+1$
      blocks
    • compute XCBC-MAC $C$ of $M_{\mathsf{start}}$ with keys $K, K_2, K_3$
    • let $\text{lsb}_j$ be least significant bit of intermediate value $C_{\lfloor j \cdot \text{width} \rfloor + 1}$
      of XCBC-MAC computation, $j = 1, 2, \ldots, B/2$; let $\text{lsb} = \text{lsb}_1|| \ldots ||\text{lsb}_{B/2}$
    • compute $e := E_K(K_{\text{ENC}} \oplus C) \oplus \text{lsb}||\mathsf{start}$
    • compute CBC-MAC $\sigma$ for $C||e||\,|M|$ with function $E_K(K_{\text{CBC}} \oplus \cdot)$,
      where $|M|$ is encoded as $B$ bit string
    • return $(C, e, \sigma)$

– Progressive Verification of $M$ and putative MAC $(C, e, \sigma)$:
    • verify CBC-MAC $\sigma$ for message $C||e||\,|M|$ with function $E_K(K_{\text{CBC}} \oplus \cdot)$;
      if verification fails then return REJECT
    • compute $\text{lsb}||\mathsf{start} := E_K(K_{\text{ENC}} \oplus C) \oplus e$
    • compute $M_{\mathsf{start}}$ from $M$ as for MAC computation, using offset $\mathsf{start}$
    • verify MAC:
        * compute XCBC-MAC of $M_{\mathsf{start}}$ with keys $K, K_2, K_3$
        * if during this computation the least significant bit of some intermediate
          value $C_{\lfloor j \cdot \text{width} \rfloor + 1}$ does not match the previously decrypted value $\text{lsb}_j$
          then stop and output REJECT
        * finally verify that the computed XCBC-MAC matches the given one $C$;
          if not then output REJECT
```

**Fig. 4.** Progressively verifiable XCBC-MAC

*Design Principle.* The idea of the construction is as follows. Using a CBC-type of MAC for $M$ ensures that, if the adversary tampers a single block for a valid massage-MAC pair, then after this incorrect block is processed, *all* subsequent intermediate results will be essentially independent of the previously computed values. This is also referred to as the error propagation property of CBC-like MACs. In particular, this means that the least significant bits of these intermediate results are likely to be independent from the encoded ones. Hence, soon after we process a fallacious block we will find the error with probability 50%, and with probability 25% after an additional check block and so on.

Starting at a random position with the computation and verification ensures that the adversary remains oblivious about the order of the blocks and thus cannot deliberately tamper the block which will be processed last. MACing the length of the message together with $C$ and $e$ basically prevents attacks in which the adversary appends an extra block to messages, e.g., if the adversary appends a block $11\ldots1$ to the message $m = 11\ldots1||\ldots||11\ldots1$ such that the verifier will not access a truly tampered block during computations. Similarly, prepending the value $\mathsf{start}$ to the rotated message prevents that two cyclically shifted messages (say, $00\ldots0||11\ldots1$ and $11\ldots1||00\ldots0$) of the same length are accidently rotated to the same message (e.g., to $00\ldots0||11\ldots1$).

The following theorem says that the cumulative detection probability grows linear with the number of accessed blocks. Hence, the detection probability is roughly uniformly distributed. Furthermore, the proof of the theorem also shows that the MAC is unforgeable in the classical sense.

**Theorem 1.** *PIR-XCBC-MAC is a $(t, \boldsymbol{q}, \boldsymbol{b})$-progressively verifiable MAC with respect to order* $\mathsf{V.nonad}$. *The cumulative detection probability is given by*

$$\Delta^{\mathsf{V.nonad}}_{(t,\boldsymbol{q},\boldsymbol{b})}(p) = p - \frac{4}{B} - \mathsf{bad}$$

*where*

$$\mathsf{bad} \leq \boldsymbol{Adv}^{prp}_{(t',q')} + \frac{18.5(q_{\mathrm{MAC}} + q_{\mathcal{V}})^2 \ + \ 6(q_{\mathrm{MAC}} + q_{\mathcal{V}})^2(b_{\mathrm{MAC}} + b_{\mathcal{V}} + 5)^2}{2^B} + \frac{1}{b_{\mathcal{V}}^L}$$

*for the advantage $\boldsymbol{Adv}^{prp}_{(t',q')}$ of any adversary running in time $t' = t + \mathcal{O}(BS)$ and making at most $q' = q_{\mathrm{MAC}}(b_{\mathrm{MAC}} + 5) + q_{\mathcal{V}}(b_{\mathcal{V}} + 5)$ queries to distinguish the block cipher $E$ from a truly random permutation.*

The bound $\Delta^{\mathsf{V.nonad}}_{(t,\boldsymbol{q},\boldsymbol{b})}(p)$ essentially stems from the fact that the adversary is oblivious about our random start position. Hence, the verification algorithm usually approaches an incorrect block after processing a fraction of $\frac{m}{n}$ blocks. Then it takes a few more check blocks to detect the error, namely, 2 on the average which are processed after another fraction of $\frac{2\mathrm{width}}{n} \approx \frac{4}{B}$ blocks. The amount $\mathsf{bad}$ originates from the (in)security of the deployed cryptographic schemes. Conclusively, the average number of blocks is roughly $50\% + 4/B$. The verifier thus reads on the average about 53% of an invalid submission for $B \geq 128$.

The proof as well as an alternative construction is omitted for space reasons.

## 4 Discussion

Our solutions show that progressive verification can be achieved. We have focused on message authentication and presented reasonable constructions of pv-MACs. Yet, some open problems remain, both in the field of MACs and other cryptographic primitives. In this section we give some possible future directions.

*Improved Solutions for pv-MACs.* We have already observed that our model of progressive verification can be varied, say, by allowing the adversary to adaptively decide upon the message based on the verifier's order. Finding good solutions for such cases is still open. Similarly, all our constructions rely on (non-adaptively) chosen orders by the verifier. Coming up with nontrivial solutions for random or adversarial orders is a challenging task.

Another interesting extension is to investigate the relationship between the number of incorrect and accessed blocks. For example, in our model the quality of the progressive verification procedure is measured against adversaries that need to tamper only a single block in a given message. This immmediately yields a lower bound of 50% for the average number of block accesses. It would be interesting to see

– how our constructions perform in the case that there are more incorrect blocks, possibly distributed over the message according to some specific distribution, and
– if better solutions can be found if the number of fallacious blocks must exceed a certain lower bound; this may be especially interesting for parallelizable computations of MACs where the "avalanche" effect of CBC-MACs disappears.

*Progressive Verification for other Primitives.* Concerning other cryptographic primitives we remark that one can transfer the solutions to the hash-and-sign principle. Namely, assume that an iterated hash functions like SHA or RIPEMD is applied to the message. These hash functions process the message similarly to CBC-MACs, and we can output the least significant bits of check blocks in addition the hash value. Then we sign the actual hash value as well as the extra output, e.g., with a number-theoretic signature function. The complete signature is given by the output of the signature function and the additional vector.

What is the advantage of the constructions in this case? Assume that the message is large such that the signing time is dominated by the hash evaluation. Say that the adversary is supposed to change one of the first blocks. Then the adversary can in fact bias the intermediate values by choosing the message blocks adaptively. In fact, she will be able to find a message such that the progressive verification procedure has to re-compute at least most parts of the hash computation. Nevertheless, in order to fool the verifier drastically the adversary has to invest some work first. This leads to a more balanced workload between the adversary and the verifier. For example, this may have applications for protection against denial-of-service attacks.

Finally, we remark that it would also be interesting to find nontrivial signature schemes in which the underlying number-theoretic function is somewhat progressively verifiable.

## Acknowledgment

We thank the anonymous reviewers for their comments.

## References

1. M. Bellare, O. Goldreich, S. Goldwasser: Incremental Cryptography: The Case of Hashing and Signing, *Advances in Cryptology — Proceedings of Crypto '94, Lecture Notes in Computer Science, Vol. 839, pp. 216–233, Springer-Verlag*, 1994.
2. M. Bellare, O. Goldreich, S. Goldwasser: Incremental Cryptography and Application to Virus Protection, *Proceedings of the 27th ACM Symposium on Theory of Computing (STOC), pp. 45–56*, 1995.
3. M. Bellare, R. Guerin, P. Rogaway: XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Funtions, *Advances in Cryptology — Proceedings of Crypto '95, Lecture Notes in Computer Science, Vol. 963, pp. 15–29, Springer-Verlag*, 1995.
4. M. Bellare, J. Kilian, P. Rogaway: The Security of Cipher Block Chaining Message Authentication Code, *Advances in Cryptology — Proceedings of Crypto '94, Lecture Notes in Computer Science, Vol. 839, pp. 341–358, Springer-Verlag*, 1994.
5. J. Black, S. Halevi, H. Krawczyk, T. Krovetz, P. Rogaway: UMAC: Fast and Secure Message Authentication, *Advances in Cryptology — Proceedings Crypto '99, Lecture Notes in Computer Science, vol. 1666, pp. 216–233, Springer-Verlag*, 1999.
6. J. Black, P. Rogaway: CBC MACs for Arbitrary-Length Messages: The Three-Key Construction, *Advances in Cryptology — Proceedings of Crypto 2000, Lecture Notes in Computer Science, Vol. 1880, pp. 197–215, Springer-Verlag*, 2000.
7. J. Black, P. Rogaway: A Block-Cipher Mode of Operation for Parallelizable Message Authentication, *Advances in Cryptology — Proceedings of Eurocrypt 2002, Lecture Notes in Computer Science, Vol. 2332, pp. 384–397, Springer-Verlag*, 2002.
8. U. Feige, A. Fiat, A. Shamir: Zero Knowledge Proofs of Identity, *Journal of Cryptology, Vol. 1, pp. 77–94, Springer-Verlag*, 1988.
9. M. Fischlin: Incremental Cryptography and Memory Checkers, *Advances in Cryptology — Proceedings of Eurocrypt '97, Lecture Notes in Computer Science, Vol. 1233, pp. 393–408, Springer-Verlag*, 1997.
10. M. Fischlin: Lower Bounds for the Signature Size of Incremental Schemes, *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 438–447, IEEE Computer Society Press*, 1997.
11. M. Fischlin: Fast Verification of Hash Chains, *to appear in RSA Security 2004 Cryptographer's Track, Lecture Notes in Computer Science, Springer-Verlag*, 2004.
12. L.C. Guillou, J.-J. Quisquater: A Practical Zero-Knowledge Protocol Fitted to Security Microprocessors Minimizing Both Transmission and Memory, *Advances in Cryptology — Proceedings of Eurocrypt '88, Lecture Notes in Computer Science, Vol. 330, pp. 123–128, Springer-Verlag*, 1988.
13. P.C. Kocher: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems, *Advances in Cryptology — Proceedings of Crypto '96, Lecture Notes in Computer Science, Vol. 1109, pp. 104–113, Springer-Verlag*, 1996.
14. C.P. Schnorr: Efficient Signature Generation by Smart Cards, *Journal of Cryptology, Vol. 4, pp. 161–174*, 1991.