ALPS: Authenticating Live Peer-to-Peer Streams

Remo Meier, Roger Wattenhofer (Regular Submission)
{remmeier,wattenhofer}@tik.ee.ethz.ch
ETH Zurich, Switzerland

Abstract

Live streaming is one among many applications where data is continuously created and has to be quickly distributed among a large number of users. The peer-to-peer paradigm is thereby attracting interest with the prospect of overcoming scalability issues of more centralized approaches. Since data blocks travel along multiple (possibly malicious) peers, authenticating the origin of blocks becomes of prime importance to guarantee safety and reliability. The asymmetry of a single source and an arbitrary number of untrusted receivers requires the use of digital signatures and public key cryptography in general. This paper proposes a new signature scheme for broadcast authentication tailored towards peer-to-peer systems to overcome limitations of traditional approaches based on signature schemes like RSA and DSA, most notably in terms of delays, signature size, and computational complexity. It may further be of practical interest for other real-time application such as massive multiplayer peer-to-peer gaming.

Keywords: Peer-to-Peer, Security, Signatures, Live Streaming

1 Introduction

As everybody knows the peer-to-peer paradigm can be applied to more than just file sharing. Recently, there is a trend towards peer-to-peer applications that need to deliver data in real-time, examples of such applications include live streaming systems and massive multiplayer games. Due to the possible presence of malicious peers, it becomes vital to provide security to all participating peers. In particular, all data blocks of a peer-to-peer live stream need to be authenticated by each peer such that fraudulent blocks injected by intermediate peers can be detected.

In applications such as file-sharing where all data blocks to be distributed are available at the beginning authentication is pretty straightforward; essentially a peer is able to authenticate the data by first retrieving a signature — normally a small list of hash values, one for each block of the data — from the source through a secure channel.

In contrast, real-time applications such as multiplayer games and live streaming face the challenge of authenticating data that is just being created. If the peer-to-peer paradigm is used to distribute the data there is an additional catch: intermediate peers must authenticate data blocks before forwarding them to other peers, preventing a ruinous snowball effect of proliferating fraudulent packets, which could waste precious bandwidth.

Commonly used signatures schemes are based on mathematical problems such as the hardness of factoring integer numbers or the hardness of computing discrete logarithms,

where the factorization or the discrete logarithm respectively acts as a trap-door to sign data blocks and is kept secret as private key. The disadvantages are a high computational complexity and large signatures to counter ever more sophisticated attacks. Clearly, such schemes do not allow to sign data blocks fitting into a network's maximum transfer unit of typically 1,500 bytes. But also amortizing the signature costs over multiple blocks is not an option for peer-to-peer systems. For example, compiling and distributing signed hash lists more than doubles delays. And a larger block size imposes more delays with each hop. Furthermore, neither of the solutions is applicable to massive multiplayer peer-to-peer games with even more stringent timing constraints.

Instead, we examine signature schemes based on arbitrary one-way functions without trap-doors [1, 2] that have been studied over the past 30 years. We propose several new techniques that extend existing one-way signature schemes such that they can be used for live peer-to-peer applications. Existing schemes either need a public key which is several hundred kilobytes long, or need all available processing power to manage authentication only, or waste a too large part of each packet for authentication. Our signature scheme $\mathcal{ALPS}$ can tradeoff the three main properties public key size, signature size, and computational delay. For instance, signatures are between 20 to 40 bytes in size. Signing and verification typically takes between merely 10 and 300 μs. And a public key a few kilobytes in size is sufficient to authenticate a live stream. Overall, $\mathcal{ALPS}$ is about two orders of magnitude more efficient than previous schemes. We believe that these two order of magnitude are necessary to make low-delay and authenticated live peer-to-peer streams practically possible.

We further incorporated $\mathcal{ALPS}$ into Pulsar[1], an open effort for a peer-to-peer streaming platform. Pulsar supports both live and on-demand streaming, and while it is still an ongoing research effort, an initial version is ready for testing and real-world use. In case of live streaming, one of the primary objectives of Pulsar is to reduce propagation delays compared to other peer-to-peer streaming systems. $\mathcal{ALPS}$ thereby plays a vital role as it allows to work with small blocks of data that can be quickly authenticated and forwarded.

The remainder of this paper is organized as follows. After reviewing related work in Section 2, the design of our signature scheme $\mathcal{ALPS}$ is presented in Section 3, followed by an evaluation in Section 4 and a conclusion in Section 5.

2 Related Work

A large number of signatures schemes have been proposed over the years. We quickly review schemes relevant to this work. Message authentication codes (MAC) allow to efficiently authenticate data with a secret key shared between sender and receiver. Unfortunately for our case, shared secret keys allow receivers to sign packets as well, rendering MACs unsuitable for applications where receivers are not trustworthy. In contrast, signature schemes based on asymmetric cryptography distribute a public key among all receivers, but keep a private key secret at the source. RSA, DSA, and ECDSA may be the most prominent schemes of this kind. But, computational complexity and signature size do not allow to sign a large number of small data blocks, for example, 1,500 bytes in size to fit into network packets. McEliece [11] is an alternative signature scheme featuring signatures merely 87 bits in size. Unfortunately,

[1] www.getpulsar.com

again, the computational costs of several seconds to sign and verify data blocks prevent its use in the peer-to-peer context. Similarly, Quartz [10] creates signatures with 128 bits in size and is simple to verify, but it also takes seconds to sign data blocks.

For this reason, a large body of work, e.g., [15, 14, 13, 16], focuses on amortizing signature costs over a number of network packets while maintaining, for example, security in case of packet loss. Such schemes have to either work with large blocks or assume an offline setting where all blocks are known from start to the source. But as argued in the introduction, either case is not an option for peer-to-peer live streaming since data is continuously created and has to be authenticated before forwarding. But authenticating a block also requires the complete download of the block, which in turn leads to more delays at each hop the larger a blocks is.

Canetti et al. proposed in [17] a scheme based on multiple MACs that avoids complex signature schemes. A block is signed with all MACs, but each receiver does not hold all, but only a subset of the corresponding secret keys. This way, a receiver is no longer able to sign blocks for other receivers if subsets of secret keys are carefully assigned to receivers. Unfortunately, the approach becomes impractical for a large number of receivers and possible collusions among them. A solution a long the same line is Tesla [12]. A MAC authenticates a data block, but its secret key is revealed only later after the packet has been distributed (based on commitments to secret keys and hash chains). Unfortunately, Tesla is incompatible with the peer-to-peer paradigm since peers could no longer authenticate blocks before forwarding them.

Signatures schemes like RSA, DSA, and ECDSA make use of complex mathematical assumptions to obtain a trap-door as private key. Signatures based on arbitrary one-way functions without the need for trap-doors have been proposed by Lamport in 1979 [1]. Compared to other signature schemes, less mathematical assumptions are needed, which in turn allows the use of simple and efficient functions like MD5 and SHA-1. Drawbacks are large keys and the availability of a single signature that also limits signing to a single block. Merkle introduced the hash tree [2] to allow the use of multiple Lamport instances to sign more than one block. Subsequent work further revised and generalized this idea. For example, a generalization to directed acyclic graphs is given in [4] and an optimal tree-based scheme in [3]. However, the interest has mostly been of theoretical nature as signature size and key size, as well as the limited number signatures generally prevented their use in practice.

A new path relevant to this work has been taken with the introduction of the *BiBa* signature scheme [5] by Perrig in 2001. *BiBa* is tailored towards broadcast authentication, i.e., verifying the origin of a sequence of packets. A signer thereby keeps a large number of hash chains secret, and includes in every new signature the next unused values of a few carefully selected chains. There are different flavors of the protocol. *Biba* itself is based on the birthday paradox by finding k-way collisions to create signatures with k values. The *Powerball* signature scheme [7] generalizes *Biba* by working with more general patters besides collisions. And the *Hors* signature scheme [6] selects chains directly using a secure hash function.

The problem of all three, *Biba*, *Hors*, and *Powerball* is the number of chains. A large number of chains also yields a large public key. And, more importantly, verifiers have at all time to be aware of how many times each chain has been used so far. Otherwise, an attacker is able to forge signatures for new blocks by

reusing parts of old signatures. In peer-to-peer systems, where peers are inherently unreliable, it can prove difficult to maintain this kind of synchronization for a large number of chains. For the same reasons, an instance can only be used a few times during the propagation delay of the overlay. Consequently, multiple instances might have to be used, further increasing the size of the public key. Moving forward in *all* chains when signing a signature avoids synchronization problems [5], but wastes most items in the chains and increases the computational complexity as verifiers also have to skip the unused items.

3 Signature Scheme

In this section, the $\mathcal{ALPS}$ signature scheme is proposed to authenticate sequences of data blocks, e.g., a peer-to-peer live streams. $\mathcal{ALPS}$ produces short signatures with low computational complexity; it addresses the issues of large public keys and synchronization problems encountered in other protocols such as Biba [5]. The $\mathcal{ALPS}$ scheme is composed of three algorithms, a key generation, a signing, and a verification algorithm. An evaluation along with a few practical extensions is given in the subsequent Section 4.

The private and public key generation works as follows. Let q denote a security parameter and $H : \{0,1\}^q \rightarrow \{0,1\}^q$ a secure hash function. The secret key consists of n uniformly at random chosen secret values $s_{1,1,1},\ldots,s_{n,1,1}$ with $s_{i,1,1} \in \{0,1\}^q$ for $i = 1,\ldots,n$. Each secret value $s_{i,1,1}$ is the starting point of a hash chain of length $bl+1$ with parameters l and b. Each chain is thereby partitioned into blocks, we have l blocks of size b, followed by a last block of size 1. An item $s_{i,j,k}$ in a chain, also referred to as *seal* [5], is addressed by chain index i, block index j, and offset k within the block. Within any chain i and block j it holds that $s_{i,j,k+1} = H(s_{i,j,k})$ for $k = 1,\ldots,b-1$. Between block j and block $j+1$ it holds that $s_{i,j+1,1} = H(s_{i,j,b})$. The last value of each chain becomes the public key, i.e., $s_{i,l+1,1}$ for $i = 1,\ldots,n$. An example is given in Figure 1.

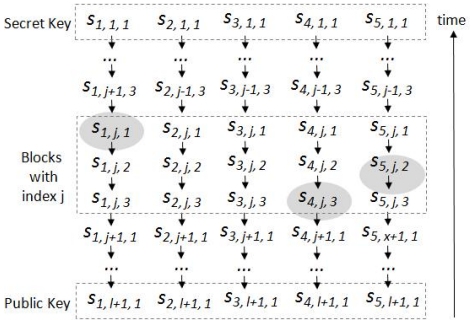

Figure 1: $\mathcal{ALPS}$ hash chains: Example with $n = 5$ chains, a signature size $\lambda = 3$, and a block size $b = 3$. One out of 70 possible signatures is highlighted. Note that in general seals of a signature may use different blocks in different chains.

A message m is signed by carefully including seals of a subset of chains in the signature. A signature is thereby encoded by both the seals' chains and offsets within the blocks. Note that the protocol is randomized and may fail with non-negligible probability. A counter c allows to perform multiple trials until the signature generation succeeds. Indices $u_1,\ldots,u_n$ are initially set to l; they determine the next available blocks of the chains $1,\ldots,n$. The protocol works in three steps, chain selection, offset selection, and the final signature generation, as follows.

The number of seals included in a signature is given by λ. In the first step, a secure hash value of the message m and counter c is used to select a distinct set of chains $\{c_1,\ldots,c_\lambda\}$ with $c_i \in \{1,\ldots,n\}$, denoted as $\mathrm{H}^{\lambda,distinct}_{\{1,\ldots,n\}}(m, c)$. From each of the λ selected chains, a seal will be included in the signature.

In the second step, a block offset $p_i \in \{1,\ldots,b\}$ is selected for each selected chain c_i. As a security precaution, the offsets p_i need

to sum up to a constant $t = \frac{1}{2}\lambda(1+b)$. Constant t corresponds to the expected value of a sum of offsets chosen uniformly at random and thereby maximizes the number of possible offset combinations. The first $\lambda - 1$ offsets are again chosen by a secure hash function, denoted as $\mathrm{H}^{\lambda-1}_{\{1,\ldots,b\}}(m,\ c,\ c_1,\ldots,c_\lambda)$. The selection $\{c_1,\ldots,c_\lambda\}$ from the first step is added as input to the hash function along with the message m and the counter c as a further security precaution. Note that in contrast to chains, offsets do not have to be distinct. The remaining offset p_λ is chosen such that the offsets sum up to constant t. Since offsets are strictly non-negative and smaller or equal to the block size b, the selection may fail to sum up. In this case, the counter c is increased by 1 and the algorithm restarts in the first step.

In the third and last step, the signature is created. The chains $\{c_1,\ldots,c_\lambda\}$ and block offsets $\{p_1,\ldots,p_\lambda\}$ have been chosen in the first and second step. The block indices are given by the indices $\{u_{c_1},\ldots,u_{c_\lambda}\}$. The corresponding seals $s_{c_1,u_{c_1},p_1,\ldots,s_{c_\lambda,u_{c_\lambda},p_\lambda}}$ and the counter c constitute the signature for message m. Each of the indices $\{u_{c_1},\ldots,u_{c_\lambda}\}$ is then decremented by one to point to the next unused block. If any of the chains has been used and no further blocks are available, a new public and secret key has to be generated. The final algorithm for signing a message is summarized by Algorithm 1.

Verifying the signature of a message works in a similar fashion as signing a message. The message and the counter included in the signature determine a selection of chains and block offsets. For each seal included in the signature, a verifier follows the hash chain until a known seal is reached, either from the public key or from previous signatures. If no known seal is found after $2b$ steps or a preceding seal is known for any of the chains, the signature is considered to be invalid. Otherwise, the number of steps determines the block offsets of the seals. If they match the ones selected by the message and the counter, then the signature is considered to be valid, and the message can be forwarded.

Algorithm 1 $\mathcal{ALPS}$

```
1: var u_1,...,u_n := l
2:
3: function sign(m)
4:   c := 0
5:   while not finished do
6:     {c_1,...,c_λ} := H^{λ,distinct}_{{1,...,n}}(m, c)
7:     {p_1,...,p_{λ-1}} := H^{λ-1}_{{1,...,b}}(m, c, c_1,...,c_λ)
8:     s = sum{p_1,...,p_{λ-1}}
9:     t = (λ*(1+b))/2
10:    if s < t and s >= t - b then
11:      p_λ := t - s
12:      sig := {s_{c_1,u_{c_1},p_1,...,s_{c_λ,u_{c_λ},p_λ},c}
13:      for i = 1,...,λ do
14:        u_{c_i} := u_{c_i} - 1
15:      od
16:      return sig
17:    fi
18:    c = c + 1
19:  od
20: end function
```

4 Evaluation

In this section, the $\mathcal{ALPS}$ signature scheme is evaluated in respect to security, computational complexity, signature size, and public key size. We further outline a few extensions that have proven useful for the adoption in the Pulsar streaming system.

4.1 Security

$\mathcal{ALPS}$ is kind of a one-time signature scheme as can be used at most once during the propagation delay of the peer-to-peer network. Once a block is signed, the source of a stream has to wait until the block reached all peers [2]. Other-

[2]Multiple independent $\mathcal{ALPS}$ instances allow to sign multiple blocks during the propagation delay.

wise, peers may not be aware of the boundary of used and unused blocks. The security provided by $\mathcal{ALPS}$ stems from the fact that the source holds all seals of all chains as secret key, while an attacker is limited to seals from previous signatures and subsequent seals in the respective chains. If peers are aware of the boundary of used and unused blocks, it limits an attacker further to the seals of the most current signature and subsequent seals of the respective blocks[3]. Since offsets of seals in a signature sum up to constant t, it is not possible that, for example, an unfortunate block selects the first seal of each block, exposing all subsequent seals in the respective blocks and limiting security to the selection of chains. Furthermore, an attacker is not able to replace any of the seals of the current signature by a subsequent seal since it would force the use of a preceding (secret) seal in another chain, considered to be computationally infeasible. Consequently, a forged signature would have to select exactly the same chains and offsets, considered to be computationally infeasible for a sufficient number of chains and a sufficient block size if the selection is performed by secure hash functions. Therefore, the security provided by $\mathcal{ALPS}$ is given by the number of possible signatures, determined by the number of chains n, signature size λ, and block size b.

4.2 Selection of a one-way function

Critical to the security and performance of $\mathcal{ALPS}$ is the selection of an appropriate one-way function to compute the hash chains. The function has to be one-way to provide robustness against preimage attacks. Otherwise, an attacker may simply invert the hash chains, depriving a signer of his advantage of being the sole peer knowing all seals. The selected function further has to withstand second preimage attacks. A second preimage attack would enable an attacker to compute a new block that has the same signature as an given existing block. But note that in contrast to secure hash functions, there is no need for collision resistance, which is considered to be more difficult to achieve. In $\mathcal{ALPS}$, a source could sign two blocks with the same signature by finding a collision, but this neither breaks authenticity nor is it of any use to an attacker.

Pulsar currently uses MD5 because of its low computational complexity and robustness against preimage and second primage attacks. While there are known attacks for MD5 speeding-up the search for second preimages in very large messages [8] and collisions [9], it is of no concern for Pulsar and $\mathcal{ALPS}$ as argued before and because data blocks are small. The design of $\mathcal{ALPS}$ allows further optimizations. For example, when following hash chains, the output of an iteration is used as input in the next iteration, allowing to keep values within CPU registers. Furthermore, it is possible to follow different hash chains concurrently, allowing the use of both multi-core processors and special instructions like *Streaming SIMD Extensions* (*SSE*). *SSE* allows to perform 128-bit vector operations and is readily available in personal computers since 2001. In case of *MD5*, based on 32-bit operations, *SSE* allows to compute four hash values simultaneously. This way, a C++ implementation running on a 2.66 GHz processor is able to perform 16 million hash operations per second with a *single* processing core.

4.3 Randomized Signing

The signature generation of $\mathcal{ALPS}$ is randomized and may fails with non-negligible probability. To determine the failure probability, a normal distribution can approximate the distribution of the first $\lambda - 1$ offsets' sum

[3] If a peer receives a signature first, an attacker has no seals at all to work with.

$p_1 + \ldots + p_{\lambda-2}$. Expected value μ and variance σ^2 are thereby given by:

$$\begin{aligned} \mu &\approx \frac{1}{2} b(\lambda - 1) \\ \sigma^2 &\approx \frac{1}{12} b^2(\lambda - 1) \end{aligned}$$

In order for all offsets, including $p_{\lambda-1}$, to sum up to t, the sum is allowed to deviate no more than $b/2$, leading to an estimate for the failure probability p_{fail}:

$$p_{fail} \approx 1 - \Phi_{\mu,\sigma^2}(\mu + \frac{b}{2}) + \Phi_{\mu,\sigma^2}(\mu - \frac{b}{2})$$

Fortunately, the failure probability p_{fail} varies between 22% for signatures with $\lambda = 3$ seals and 56% with $\lambda = 10$ seals. In any case, this randomness introduces only a minor computational overhead.

4.4 Signature Size

Our parameters n, b, λ, q provide trade-offs between computational cost, public key size, signature size, and security. In case of peer-to-peer live streaming or gaming, a moderate security level between 2^{50} and 2^{60} is usually sufficient since data blocks expire within few seconds. For example, Pulsar allows to distribute data block in one to two seconds among millions of peers if sufficient bandwidth is available. As argued in Section 4.1, the number of different $\mathcal{ALPS}$ signatures $\mathcal{S}$ essentially determines the security level. Failure probability p_{fail} from the previous section thereby allows to approximate $\mathcal{S}$:

$$\mathcal{S} \approx (1 - p_{fail}) b^{\lambda-1} \binom{n}{\lambda}$$

For example, $n = 50$ chains, $\lambda = 5$ seals in a signature, and block size $b = 489$ lead to $\mathcal{S} = 2^{56}$. The verification time t_{verify} is mainly given by the number of one-way function evaluation, i.e. the product of λ and b. In the given example, MD5 has a verification time $t_{verify} = 153$ μs. Table 2 depicts the verification times of more examples for fixed $\mathcal{S}$, n, and λ. In any case, the low computational complexity allows to quickly authenticate a large number of blocks.

$\mathcal{S}$	λ	n=25	n=50	n=100	n=200
2^{48}	4	805	306	119	47
	5	96	38	16	7
	6	30	12	5	4
2^{56}	4	5,111	1,943	755	297
	5	383	153	63	26
	6	89	36	15	7
2^{64}	5	1,533	610	250	104
	6	269	110	46	19

Figure 2: $\mathcal{ALPS}$: Verification time in μs for a given security level $\mathcal{S}$, number of chains n, and signature size λ.

λ=n	b	t_{verify}
5	4,656	1,455 μs
6	875	328 μs
7	287	126 μs
8	129	64 μs
9	71	40 μs
10	45	28 μs

Figure 3: $\mathcal{ALPS}$: Block size b and verification time t_{verify} required to reach a security level of 2^{48} if the signature size λ matches the number of chains n.

Of special interest is the case where the number of chains n matches the signature size λ. Figure 3 shows the block size and verification times required to reach security level $\mathcal{S} = 2^{48}$ for different signature sizes λ. Naturally, either more seals in signatures or a larger block size is required to compensate for the lack of flexibility in selecting chains. In contrast, public keys become small and synchronization is limited to knowing the current block index used by all chains.

The size of signatures is given by the number of seals λ and the seal size. While MD5 out-

puts 128-bit values, between 48 to 72 bits are sufficient if data blocks expire quickly and robustness agains pre-image attacks is given. In case of $\lambda = 5$ and $\mathcal{S} = 2^{56}$, the signature size is 35. Some care has to be taken since hash chains have a lifetime from several minutes to hours. Adding an additional chain, known as salt chain, can resolve the problem. To then compute a seal in a seal chain, the salt is added as input, along with the previous seal in the respective chain, to the one-way function. We refer to [5] for a more detailed discussion about salt chains.

Typically, only intermediate seals of each chain are stored by the signer to save space, increasing the computational complexity of signing as the signer has to follow hash chains as well. If, for example, the seal at the beginning of each block is stored, then the signing time is about half the verification time[4].

4.5 Comparision to other schemes

To compare $\mathcal{ALPS}$ with more commonly used signature schemes, namely RSA, DSA, and ECDSA, their computational complexity to sign and verify messages and their signature sizes are depicted in Figure 4. The tests have been performed with Java 6 and the same 2.66 GHz processor. While RSA signatures are quickly verified, signing is computationally expensive and, more importantly, the signature size is too large to sign network packets with a typical MTU of 1,500 bytes. In contrast, DSA features small signatures but incurs high computational costs. And finally, ECDSA, based on elliptic curve cryptography, allows the use of shorter keys, but is computationally the most expensive of the three schemes.

Biba [5], *Powerball* [7], and *Hors* [6] are signature schemes similar to $\mathcal{ALPS}$. While these schemes allow signing of multiple data blocks during the propagation delay, it is not a disadvantage for $\mathcal{ALPS}$ for two reasons. First, $\mathcal{ALPS}$ makes use of smaller public keys, which in turn allows the use of more instances. Second, the security of *Biba*, *Powerball*, and *Hors* decreases exponentially with every signed block. For example, signing four blocks with $\lambda = 6$ seals each gives an attacker about 2^{17} possibilities to select seals for its own, forged signature. Assuming that $1,000$ chains are used, another five seals would be needed to maintain the same security level as the additional seals further boost the number of possibilities to 2^{33}. Taking the example from the previous section, *Biba* requires the use of $6,000$ chains to maintain a security level $\mathcal{S} = 2^{56}$ with $\lambda = 5$ seals if it is used once during the propagation delay. This is highly undesirable both for public key distribution and for keeping peers up-to-date about the boundary of used and unused seals. The later issue can be solved by moving forward in all chains with every signature, as noted in Section 2, but in this case *Biba* further loses its performance advantage over $\mathcal{ALPS}$ while still having a large public key. *Powerball* and *Hors* perform similar to *Biba*. In any case, the performance of $\mathcal{ALPS}$ is more than adequate to authenticate a large number of small blocks.

	Key	Signature	t_{sign}	t_{verify}
DSA	1024	376	4064	8875
RSA	1024	1024	6150	324
ECDSA	256	568	23006	30409

Figure 4: Key and signature size (bits) and computational complexity (μs) for RSA, DSA, and ECDSA.

4.6 Propagation delays

Signing times in $\mathcal{ALPS}$ are short. A way to improve security at the cost of (slightly) longer signing times is to increase the computational

[4]It is possible to implement $\mathcal{ALPS}$ without fixed blocks, also reducing the verification time by a factor of two at the cost of a more difficult synchronization.

complexity for signing blocks. While the impact is small for a honest signer, it becomes considerably harder for an attacker to forge a signature.

A simple solution is to lower the success probability for signing blocks. Enforcing that a signature further satisfies $H^1_{\{0,\ldots,\theta-1\}}(c,c_1,\ldots,c_\lambda) = \beta$ with $\theta = 1024$ and $\beta = 0$ decreases the success probability by factor θ for both signers and attackers.

A more sophicasted scheme may choose a non-random β. Pulsar, for example, makes use of the propagation delays. Namely, solely the source of a live stream has *all* blocks in its buffer. It takes time for blocks to reach other peers, especially since different blocks usually travel along different paths to avoid bottlenecks and single point of failures. To sign a new data block, the source hashes $log(\theta)$ preceding blocks to one bit each and sets β to their concatentation. Other peer subsequently verify the bits for preceding data blocks they already received. There is a good chance a malicious peer gets caught for sending forged blocks because its signatures are based on incomplete knowledge of preceding blocks.

4.7 Synchronizing peers

Peers have to be weakly synchronized with the source in order to be aware of the boundary of used and unused blocks. And while there are no guarantees to receive blocks in time, a variety of techniques help to maintain synchronization. For example, while insufficient bandwidth may prevent the download of blocks, peers should retrieve the much smaller signatures.

Hash lists complement $\mathcal{ALPS}$ in Pulsar to both increase security and to help keeping boundaries up-to-date. $\mathcal{ALPS}$ is thereby used during the inital distribution of new blocks until hash lists become available. Blocks are then double checked and boundaries updated if necessary. Eased security requirements for $\mathcal{ALPS}$ and larger block sizes within hash lists offset any additional overhead.

Hash lists further enable newly joined peers to gain synchronization by downloading the most recent hash lists, signatures, and either blocks or their (small) hash values. The synchronization is completed as soon as the seals in the downloaded signatures cover all chains. The well known *coupon collector problem* provides an approximation[5] for the expected number of downloads as also suggested in [5]. A peer has to download about $\frac{nlog(n)}{\lambda}$ signatures. Fewer chains in $\mathcal{ALPS}$ compared to schemes like Biba make this approach viable in practice. Using the *power of two choices* [18] gives further improvements as depicted in Figure 5. For example, with $n = 64$ chains and $\lambda = 4$ seals in a signature, a peer has to download 4.6 times as many seals as there are chains to synchronize (296 seals in total). In contrast, getting the current boundary from the source would only require the download of a single seal per chain. By balancing the use of seals in signatures, i.e., generating several signatures and choosing the one with seals from chains the least used, peers synchronize more quickly. Balancing among two signatures yields a ratio of 3.3, while 16 signatures lead to a ratio of 1.6.

5 Conclusions

Signatures schemes are ubiquitous in todays Internet. But size and computational complexity limit there use to signing larger blocks of data. In contrast, applications like peer-to-peer live streaming and peer-to-peer gaming aim at working with small data blocks to quickly forward new data. Otherwise, delays sum up with each hop in the peer-to-

[5]Note that seals within a signature are from distinct chains, leading to slightly smaller expectations compared to the coupon collector problem.

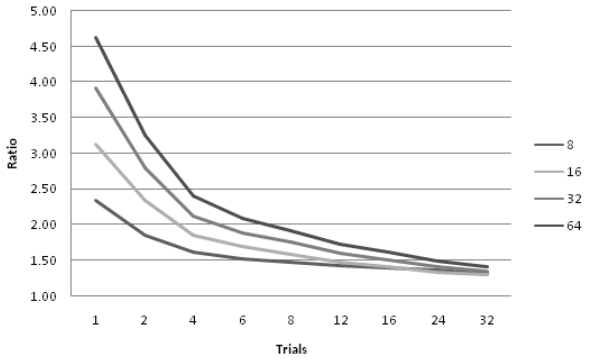

Figure 5: Expected number of seals required per chain to synchronize with 8, 16, 32 and 64 chains and balancing with 1 to 32 trials.

peer overlay. And in any case, data needs to be authenticated before forwarding. The $\mathcal{ALPS}$ signature scheme is proposed to make low-delay and authenticated peer-to-peer live streaming feasible. $\mathcal{ALPS}$ features small signatures, fast signature generation and verification, and small public keys. We further incorporated $\mathcal{ALPS}$ into Pulsar, an open and ongoing research effort for a peer-to-peer streaming platform.

References

[1] L. Lamport. Constructing digital signatures from a one way function. Technical Report CSL-98, SRI International, October 1979.

[2] R. Merkle. Secrecy, Authentication, and Public Key Systems. UMI Research Press, 1982.

[3] D. Bleichenbacher, U. Maurer. Optimal Tree-Based One-time Digital Signature Schemes. STACS'96, LNCS, Vol. 1046, Springer-Verlag, 1996.

[4] D. Bleichenbacher, U. Maurer. Directed Acyclic Graphs, One-way Functions and Digital Signatures. In Advances in Cryptology - Crypto'94, LNCS, Vol. 839, Springer-Verlag, 1994.

[5] A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In Eighth ACM Conference on Computer and Communication Security. ACM, November 5–8 2001.

[6] L. Reyzin, N. Reyzin. Better than BiBa: Short One-time Signatures with Fast Signing and Verifying. In Proceedings of 7th Australasian Conference on Information Security and Privacy, LNCS, Vol. 2384, Springer-Verlag, 2002.

[7] M. Mitzenmacher, A. Perrig. Bounds and improvements for BiBa signature schemes. Technical Report (TR-02-02), Harvard University, 2002.

[8] J. Kelsey, B. Schneier. Second Preimages on n-bit Hash Functions for Much Less than 2n Work. In Advances in Cryptology - EUROCRYPT 2005, LNCS, Vol. 3494, Springer-Verlag, 2005.

[9] X. Wang, D. Feng, X. Lai, H. Yu. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199.

[10] N. Courtois, L. Goubin, J Patarin. Quartz, 128-bit long digital signatures. Rsa Conference 2001, LNCS, Vol. 2020, Springer-Verlag, 2001.

[11] N. Courtois, M. Finiasz, N. Sendrier. How to achieve a McEliece-based Digital Signature Scheme. In Advances in Cryptology - ASIACRYPT 2001, LNCS, Vol. 2248, Springer-Verlag, 2001.

[12] A. Perrig, R. Canetti, J.D. Tygar, D. Song. The TESLA Broadcast Authentication Protocol. In Cryptobytes, Vol. 5, No. 2, RSA Laboratories, 2002.

[13] J.M. Park, E. Chong, H. Siegel. Effcient multicast packet authentication using signature amortization. In Proceedings of IEEE Symposium on Security and Privacy, 2002.

[14] S. Miner, J. Staddon. Graph-based authentication of digital streams. In Proceedings of the IEEE Symposium on Research in Security and Privacy, 2001.

[15] R. Gennaro, P. Rohatgi. How to sign digital streams. In Advances in Cryptology CRYPTO97, LNCS, Vol. 1294, Springer-Verlag, 1997.

[16] C.K. Wong, S. Lam. Digital Signatures for Flows and Multicasts. In IEEE/ACM Transactions on Networking, Vol. 7, 1999.

[17] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas. Multicast Security: A Taxonomy and Some Efficient Constructions. In IEEE Infocom, LNCS, Vol. 1294, Springer-Verlag, 1999.

[18] Y. Azar, A. Z. Broder, A. R. Karlin, E. Upfal. Balanced allocations. In SIAM Journal on Computing, Vol. 29, 2000.