ALPHA

Johannes Gilger, Florian Weingarten

January 23rd, 2009

Adaptive and Lightweight Protocol
for Hop-By-Hop Authentication

Summary

What we did since last meeting

- Finished implementing hash chain framework
- Exchange of hash chain anchors via initial handshake
- Intermediate storage model (good balance between storage and computation)
- „evil `ipqueue` filter“

Problems

- When to think of a packet as „OK“ (i.e. when to decrement the counter)
- Alpha screws up when using our „evil `ipqueue` filter“
- Alpha protocol state machine is not specified completely in paper!

Summary

What we did since last meeting

- Finished implementing hash chain framework
- Exchange of hash chain anchors via initial handshake
- Intermediate storage model (good balance between storage and computation)
- „evil ipqueue filter“

Problems

- When to think of a packet as „OK“ (i.e. when to decrement the counter)
- Alpha screws up when using our „evil ipqueue filter“
- Alpha protocol state machine is not specified completely in paper!

Summary

What we did since last meeting

- Finished implementing hash chain framework
- Exchange of hash chain anchors via initial handshake
- Intermediate storage model (good balance between storage and computation)
- „evil `ipqueue` filter“

Problems

- When to think of a packet as „OK“ (i.e. when to decrement the counter)
- Alpha screws up when using our „evil `ipqueue` filter“
- Alpha protocol state machine is not specified completely in paper!

Alpha-packets - recap

What different Alpha packets are there?

S1 new hash anchor $h_i^{S_s}$ and an HMAC of the packet to follow $M(h_{i-1}^{S_s}, m)$

A1 new hash anchor $h_i^{V_a}$ and returns received anchor $h_i^{S_s}$

S2 new hash anchor $h_{i-1}^{S_s}$ and message m

Problems

- None, just a little complex

Alpha-packets - recap

What different Alpha packets are there?

S1 new hash anchor $h_i^{S_s}$ and an HMAC of the packet to follow $M(h_{i-1}^{S_s}, m)$

A1 new hash anchor $h_i^{V_a}$ and returns received anchor $h_i^{S_s}$

S2 new hash anchor $h_{i-1}^{S_s}$ and message m

Problems

- None, just a little complex

Alpha-packets - recap

What different Alpha packets are there?

S1 new hash anchor $h_i^{S_s}$ and an HMAC of the packet to follow $M(h_{i-1}^{S_s}, m)$

A1 new hash anchor $h_i^{V_a}$ and returns received anchor $h_i^{S_s}$

S2 new hash anchor $h_{i-1}^{S_s}$ and message m

Problems

- None, just a little complex

Hash chains - Problems recap

Problem / design decision I

Q Store everything or compute everything?

A Store only every kth element. Compute everything in between as needed

Problem / design decision II

Q Different environments: Storage constraints vs. computing constraints

A Number of elements to be stored can be set during build-time

A Small hash chains can be compensated for with frequent anchor redistribution

Problem / design decision I

Q Store everything or compute everything?

A Store only every kth element. Compute everything in between as needed

Problem / design decision II

Q Different environments: Storage constraints vs. computing constraints

A Number of elements to be stored can be set during build-time

A Small hash chains can be compensated for with frequent anchor redistribution

Hash chains - Problems recap

Problem / design decision I

Q Store everything or compute everything?

A Store only every kth element. Compute everything in between as needed

Problem / design decision II

Q Different environments: Storage constraints vs. computing constraints

A Number of elements to be stored can be set during build-time

A Small hash chains can be compensated for with frequent anchor redistribution

Our hash chain data structure

0	1	2	3	4	5	6	7	8	9
$h^0(x)$	$h^{10}(x)$	$h^{20}(x)$	$h^{30}(x)$	$h^{40}(x)$	$h^{50}(x)$	$h^{60}(x)$	$h^{70}(x)$	$h^{80}(x)$	$h^{90}(x)$

- hash chain length 100, but only 10 elements in memory
- worst-case computation rounds: 10

Example: We want $h^{42}(x)$

The way it was until now:

- Compute 42 iterations of h on seed x
- Next round, we will need $h^{41}(x)$
- Compute 41 iterations again, even though we just did that before

With our new storage scheme:

- We have $h^{40}(x)$ in memory
- Compute $h^2(h^{40}(x))$, which are only two iterations

Our hash chain data structure

0	1	2	3	4	5	6	7	8	9
$h^0(x)$	$h^{10}(x)$	$h^{20}(x)$	$h^{30}(x)$	$h^{40}(x)$	$h^{50}(x)$	$h^{60}(x)$	$h^{70}(x)$	$h^{80}(x)$	$h^{90}(x)$

- hash chain length 100, but only 10 elements in memory
- worst-case computation rounds: 10

Example: We want $h^{42}(x)$

The way it was until now:

- Compute 42 iterations of h on seed x
- Next round, we will need $h^{41}(x)$
- Compute 41 iterations again, even though we just did that before

With our new storage scheme:

- We have $h^{40}(x)$ in memory
- Compute $h^2(h^{40}(x))$, which are only two iterations

Our hash chain data structure

0	1	2	3	4	5	6	7	8	9
$h^{0}(x)$	$h^{10}(x)$	$h^{20}(x)$	$h^{30}(x)$	$h^{40}(x)$	$h^{50}(x)$	$h^{60}(x)$	$h^{70}(x)$	$h^{80}(x)$	$h^{90}(x)$

- hash chain length 100, but only 10 elements in memory
- worst-case computation rounds: 10

Example: We want $h^{42}(x)$

The way it was until now:

- Compute 42 iterations of h on seed x
- Next round, we will need $h^{41}(x)$
- Compute 41 iterations again, even though we just did that before

With our new storage scheme:

- We have $h^{40}(x)$ in memory
- Compute $h^{2}(h^{40}(x))$, which are only two iterations

Our hash chain data structure

0	1	2	3	4	5	6	7	8	9
$h^0(x)$	$h^{10}(x)$	$h^{20}(x)$	$h^{30}(x)$	$h^{40}(x)$	$h^{50}(x)$	$h^{60}(x)$	$h^{70}(x)$	$h^{80}(x)$	$h^{90}(x)$

- hash chain length 100, but only 10 elements in memory
- worst-case computation rounds: 10

Example: We want $h^{42}(x)$

The way it was until now:

- Compute 42 iterations of h on seed x
- Next round, we will need $h^{41}(x)$
- Compute 41 iterations again, even though we just did that before

With our new storage scheme:

- We have $h^{40}(x)$ in memory
- Compute $h^2(h^{40}(x))$, which are only two iterations

Our hash chain data structure

0	1	2	3	4	5	6	7	8	9
$h^0(x)$	$h^{10}(x)$	$h^{20}(x)$	$h^{30}(x)$	$h^{40}(x)$	$h^{50}(x)$	$h^{60}(x)$	$h^{70}(x)$	$h^{80}(x)$	$h^{90}(x)$

- hash chain length 100, but only 10 elements in memory
- worst-case computation rounds: 10

Example: We want $h^{42}(x)$

The way it was until now:

- Compute 42 iterations of h on seed x
- Next round, we will need $h^{41}(x)$
- Compute 41 iterations again, even though we just did that before

With our new storage scheme:

- We have $h^{40}(x)$ in memory
- Compute $h^2(h^{40}(x))$, which are only two iterations

Our hash chain data structure

0	1	2	3	4	5	6	7	8	9
$h^0(x)$	$h^{10}(x)$	$h^{20}(x)$	$h^{30}(x)$	$h^{40}(x)$	$h^{50}(x)$	$h^{60}(x)$	$h^{70}(x)$	$h^{80}(x)$	$h^{90}(x)$

- hash chain length 100, but only 10 elements in memory
- worst-case computation rounds: 10

Example: We want $h^{42}(x)$

The way it was until now:

- Compute 42 iterations of h on seed x
- Next round, we will need $h^{41}(x)$
- Compute 41 iterations again, even though we just did that before

With our new storage scheme:

- We have $h^{40}(x)$ in memory
- Compute $h^2(h^{40}(x))$, which are only two iterations

Basic alpha signature scheme: *reloaded*

Basic alpha signature scheme: *reloaded*

vm1: Handshake

vm2: Handshake

Basic alpha signature scheme: *reloaded*

vm1: Handshake

```
Initiating handshake with vm2, sending SYN.
```

vm2: Handshake

Basic alpha signature scheme: *reloaded*

vm1: Handshake

```
Initiating handshake with vm2, sending SYN.
```

vm2: Handshake

```
Got SYN packet. Sending ACK.
```

Basic alpha signature scheme: *reloaded*

vm1: Handshake

```
Initiating handshake with vm2, sending SYN.
Got ACK packet. Handshake is done! Sending ACKACK.
SIGN anchor: 3197e...
 ACK anchor: 67478...
```

vm2: Handshake

```
Got SYN packet. Sending ACK.
```

Basic alpha signature scheme: *reloaded*

vm1: Handshake

```
Initiating handshake with vm2, sending SYN.
Got ACK packet. Handshake is done! Sending ACKACK.
SIGN anchor: 3197e...
 ACK anchor: 67478...
```

vm2: Handshake

```
Got SYN packet. Sending ACK.
Got ACKACK packet. Handshake with vm1 is done!
SIGN anchor: 7568d...
 ACK anchor: 70123...
```

Basic alpha signature scheme: *reloaded*

vm1: Handshake

```
Initiating handshake with vm2, sending SYN.
Got ACK packet. Handshake is done! Sending ACKACK.
SIGN anchor: 3197e...
 ACK anchor: 67478...
```

vm2: Handshake

```
Got SYN packet. Sending ACK.
Got ACKACK packet. Handshake with vm1 is done!
SIGN anchor: 7568d...
 ACK anchor: 70123...
```

vm1: Shell

```
$
```

vm1: Signature scheme

vm2: Signature scheme

Basic alpha signature scheme: *reloaded*

vm1: Handshake

```
Initiating handshake with vm2, sending SYN.
Got ACK packet. Handshake is done! Sending ACKACK.
SIGN anchor: 3197e...
 ACK anchor: 67478...
```

vm2: Handshake

```
Got SYN packet. Sending ACK.
Got ACKACK packet. Handshake with vm1 is done!
SIGN anchor: 7568d...
 ACK anchor: 70123...
```

vm1: Shell

```
$ ping vm2
PING vm2 (192.168.10.61) 56(84) bytes of data.
```

vm1: Signature scheme

vm2: Signature scheme

Basic alpha signature scheme: *reloaded*

vm1: Handshake

```
Initiating handshake with vm2, sending SYN.
Got ACK packet. Handshake is done! Sending ACKACK.
SIGN anchor: 3197e...
 ACK anchor: 67478...
```

vm2: Handshake

```
Got SYN packet. Sending ACK.
Got ACKACK packet. Handshake with vm1 is done!
SIGN anchor: 7568d...
 ACK anchor: 70123...
```

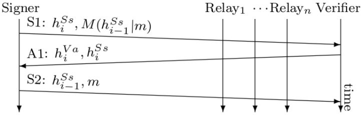
vm1: Shell

```
$ ping vm2
PING vm2 (192.168.10.61) 56(84) bytes of data.
```

vm1: Signature scheme

```
Starting signature scheme
Sending S1 (new SIGN anchor: 21f0d..., HMAC: ccfbe...)
```

vm2: Signature scheme



Basic alpha signature scheme: *reloaded*

vm1: Handshake

```
Initiating handshake with vm2, sending SYN.
Got ACK packet. Handshake is done! Sending ACKACK.
SIGN anchor: 3197e...
 ACK anchor: 67478...
```

vm2: Handshake

```
Got SYN packet. Sending ACK.
Got ACKACK packet. Handshake with vm1 is done!
SIGN anchor: 7568d...
 ACK anchor: 70123...
```

vm1: Shell

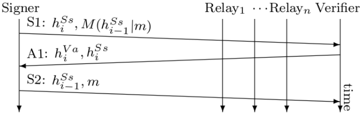
```
$ ping vm2
PING vm2 (192.168.10.61) 56(84) bytes of data.
```

vm1: Signature scheme

```
Starting signature scheme
Sending S1 (new SIGN anchor: 21f0d..., HMAC: ccfbe...)
```

vm2: Signature scheme

```
Got S1 (new SIGN anchor: 21f0d..., hash: 7568d..., expected: 7568d..., OK!)
```



Basic alpha signature scheme: *reloaded*

vm1: Handshake

```
Initiating handshake with vm2, sending SYN.
Got ACK packet. Handshake is done! Sending ACKACK.
SIGN anchor: 3197e...
 ACK anchor: 67478...
```

vm2: Handshake

```
Got SYN packet. Sending ACK.
Got ACKACK packet. Handshake with vm1 is done!
SIGN anchor: 7568d...
 ACK anchor: 70123...
```

vm1: Shell

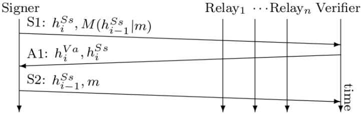
```
$ ping vm2
PING vm2 (192.168.10.61) 56(84) bytes of data.
```

vm1: Signature scheme

```
Starting signature scheme
Sending S1 (new SIGN anchor: 21f0d..., HMAC: ccfbe...)
```

vm2: Signature scheme

```
Got S1 (new SIGN anchor: 21f0d..., hash: 7568d..., expected: 7568d..., OK!)
Sending A1 (new ACK anchor: 80ec7...)
```



Signer
Relay_1 $\cdots$ Relay_n
Verifier
S1: $h_i^{Ss}, M(h_{i-1}^{Ss}|m)$
A1: h_i^{Va}, h_i^{Ss}
S2: h_{i-1}^{Ss}, m
time

Basic alpha signature scheme: *reloaded*

vm1: Handshake

```
Initiating handshake with vm2, sending SYN.
Got ACK packet. Handshake is done! Sending ACKACK.
SIGN anchor: 3197e...
 ACK anchor: 67478...
```

vm2: Handshake

```
Got SYN packet. Sending ACK.
Got ACKACK packet. Handshake with vm1 is done!
SIGN anchor: 7568d...
 ACK anchor: 70123...
```

vm1: Shell

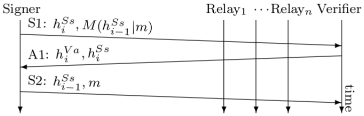
```
$ ping vm2
PING vm2 (192.168.10.61) 56(84) bytes of data.
```

vm1: Signature scheme

```
Starting signature scheme
Sending S1 (new SIGN anchor: 21f0d..., HMAC: ccfbe...)
Got A1 (new ACK anchor: 80ec7..., hash: 67478..., expected: 67478..., OK!)
```

vm2: Signature scheme

```
Got S1 (new SIGN anchor: 21f0d..., hash: 7568d..., expected: 7568d..., OK!)
Sending A1 (new ACK anchor: 80ec7...)
```


Signer
Relay$_1$ ···Relay$_n$ Verifier
S1: $h_i^{Ss}, M(h_{i-1}^{Ss}|m)$
A1: h_i^{Va}, h_i^{Ss}
S2: h_{i-1}^{Ss}, m
time

Basic alpha signature scheme: *reloaded*

vm1: Handshake

```
Initiating handshake with vm2, sending SYN.
Got ACK packet. Handshake is done! Sending ACKACK.
SIGN anchor: 3197e...
 ACK anchor: 67478...
```

vm2: Handshake

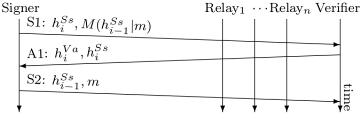
```
Got SYN packet. Sending ACK.
Got ACKACK packet. Handshake with vm1 is done!
SIGN anchor: 7568d...
 ACK anchor: 70123...
```

vm1: Shell

```
$ ping vm2
PING vm2 (192.168.10.61) 56(84) bytes of data.
```

vm1: Signature scheme

```
Starting signature scheme
Sending S1 (new SIGN anchor: 21f0d..., HMAC: ccfbe...)
Got A1 (new ACK anchor: 80ec7..., hash: 67478..., expected: 67478..., OK!)
Sending S2 with payload
```


Signer
Relay$_1$ $\cdots$ Relay$_n$ Verifier
S1: $h_i^{Ss}, M(h_{i-1}^{Ss}|m)$
A1: h_i^{Va}, h_i^{Ss}
S2: h_{i-1}^{Ss}, m
time

vm2: Signature scheme

```
Got S1 (new SIGN anchor: 21f0d..., hash: 7568d..., expected: 7568d..., OK!)
Sending A1 (new ACK anchor: 80ec7...)
```

Basic alpha signature scheme: *reloaded*

vm1: Handshake

```
Initiating handshake with vm2, sending SYN.
Got ACK packet. Handshake is done! Sending ACKACK.
SIGN anchor: 3197e...
 ACK anchor: 67478...
```

vm2: Handshake

```
Got SYN packet. Sending ACK.
Got ACKACK packet. Handshake with vm1 is done!
SIGN anchor: 7568d...
 ACK anchor: 70123...
```

vm1: Shell

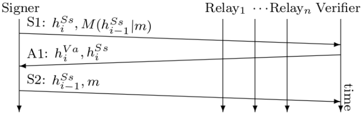
```
$ ping vm2
PING vm2 (192.168.10.61) 56(84) bytes of data.
```

vm1: Signature scheme

```
Starting signature scheme
Sending S1 (new SIGN anchor: 21f0d..., HMAC: ccfbe...)
Got A1 (new ACK anchor: 80ec7..., hash: 67478..., expected: 67478..., OK!)
Sending S2 with payload
```

vm2: Signature scheme

```
Got S1 (new SIGN anchor: 21f0d..., hash: 7568d..., expected: 7568d..., OK!)
Sending A1 (new ACK anchor: 80ec7...)
Got S2 (Stored HMAC: ccfbe..., S2 HMACed: ccfbe..., OK!)
```


Signer
Relay$_1$ $\cdots$Relay$_n$ Verifier
S1: $h_i^{Ss}, M(h_{i-1}^{Ss}|m)$
A1: h_i^{Va}, h_i^{Ss}
S2: h_{i-1}^{Ss}, m
time

Basic alpha signature scheme: *reloaded*

vm1: Handshake

```
Initiating handshake with vm2, sending SYN.
Got ACK packet. Handshake is done! Sending ACKACK.
SIGN anchor: 3197e...
 ACK anchor: 67478...
```

vm2: Handshake

```
Got SYN packet. Sending ACK.
Got ACKACK packet. Handshake with vm1 is done!
SIGN anchor: 7568d...
 ACK anchor: 70123...
```

vm1: Shell

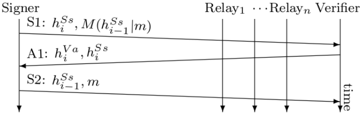
```
$ ping vm2
PING vm2 (192.168.10.61) 56(84) bytes of data.
```

vm1: Signature scheme

```
Starting signature scheme
Sending S1 (new SIGN anchor: 21f0d..., HMAC: ccfbe...)
Got A1 (new ACK anchor: 80ec7..., hash: 67478..., expected: 67478..., OK!)
Sending S2 with payload
Signature scheme done.
```

vm2: Signature scheme

```
Got S1 (new SIGN anchor: 21f0d..., hash: 7568d..., expected: 7568d..., OK!)
Sending A1 (new ACK anchor: 80ec7...)
Got S2 (Stored HMAC: ccfbe..., S2 HMACed: ccfbe..., OK!)
Signature scheme done.
```


Signer
Relay_1 $\cdots$ Relay_n
Verifier
S1: $h_i^{Ss}, M(h_{i-1}^{Ss}|m)$
A1: h_i^{Va}, h_i^{Ss}
S2: h_{i-1}^{Ss}, m
time

Basic alpha signature scheme: *reloaded*

vm1: Handshake

```
Initiating handshake with vm2, sending SYN.
Got ACK packet. Handshake is done! Sending ACKACK.
SIGN anchor: 3197e...
 ACK anchor: 67478...
```

vm2: Handshake

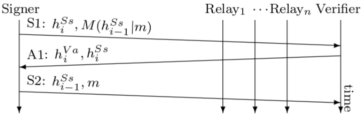
```
Got SYN packet. Sending ACK.
Got ACKACK packet. Handshake with vm1 is done!
SIGN anchor: 7568d...
 ACK anchor: 70123...
```

vm1: Shell

```
$ ping vm2
PING vm2 (192.168.10.61) 56(84) bytes of data.
64 bytes from vm2 (192.168.10.61): icmp_seq=1 ttl=64 time=2.48 ms
```

vm1: Signature scheme

```
Starting signature scheme
Sending S1 (new SIGN anchor: 21f0d..., HMAC: ccfbe...)
Got A1 (new ACK anchor: 80ec7..., hash: 67478..., expected: 67478..., OK!)
Sending S2 with payload
Signature scheme done.
```


Signer
Relay$_1$ $\cdots$Relay$_n$ Verifier
S1: $h_i^{Ss}, M(h_{i-1}^{Ss}|m)$
A1: h_i^{Va}, h_i^{Ss}
S2: h_{i-1}^{Ss}, m
time

vm2: Signature scheme

```
Got S1 (new SIGN anchor: 21f0d..., hash: 7568d..., expected: 7568d..., OK!)
Sending A1 (new ACK anchor: 80ec7...)
Got S2 (Stored HMAC: ccfbe..., S2 HMACed: ccfbe..., OK!)
Signature scheme done.
```

Goals

What we want to do next

- Figure out what to do when forged packets are detected (and implement the solution)
- Problem: high latency, figure out a way to reduce this
  64 bytes from vm2 (192.168.10.61): icmp_seq=1 ttl=64 time=2.48 ms
- Figure out a routing solution for Mac OS X
- Make alpha work on the nokia
- ...

Goals

What we want to do next

- Figure out what to do when forged packets are detected (and implement the solution)
- Problem: high latency, figure out a way to reduce this

  ```
  64 bytes from vm2 (192.168.10.61): icmp_seq=1 ttl=64 time=2.48 ms
  ```

- Figure out a routing solution for Mac OS X
- Make alpha work on the nokia
- ...