Fractal Hash Sequence Representation and Traversal

Markus Jakobsson[1]
RSA Laboratories
Bedford, MA 01730
e-mail: mjakobsson@rsasecurity.com

Abstract — **We introduce a novel amortization technique for computation of consecutive preimages of hash chains, given knowledge of the seed. While all previously known techniques have a memory-times-computational complexity of $O(n)$ per chain element, the complexity of our technique can be upper bounded at $O(log^2 n)$, making it a useful primitive for low-cost applications such as authentication, signatures and micro-payments.**

I. Introduction

With a trend towards smaller and smaller computers, there is a need for more efficient cryptographic methods. The hash chain is an elegant and versatile low-cost technique with applications ranging from renewal of certificates [4], to auctions [10], micro-payments [8] and authentication [5, 6]. All these applications have in common a need to generate consecutive hash pre-images (given some secret seed) – whether the release of such a value means that a public key has been re-certified; the bid increased; a payment performed; or a new short-term authentication key has been made available.

Let us consider one of these applications in more detail: The work on authentication [5, 6] was performed with authentication for smart dust [7] in mind. This terms refers to *very small* computational devices covering a large area, often for purposes of surveillance, whether seismic or military. We refer to [1, 11] for a more exhaustive list of applications. It is clearly the case that one wishes to conserve both computational and memory resources in such an application. However, traditional techniques of hash chain generation have complexity $O(n)$, counting the product of the amount of computation per output, and the required storage. In particular, they require either all values to be stored; computed by iterated hashing from the seed; or a straighforward hybrid of these. This largely rules out the inexpensive deployment of smart dust, and makes other applications costly as well.

We introduce a novel method with a complexity of only $O(log^2 n)$. Our method is directly applicable to authentication on "limited devices", and more generally, to light-weight applications involving the use of hash chains. In particular, our algorithm requires $\lceil log_2 n \rceil$ hash function applications per output element, and uses $\lceil log_2 n \rceil + 1$ memory cells, where each cell stores one hash chain value along with some short state information. For example, if the hash chain has length 2^{32}, and SHA [9] is the hash function of choice, then each one of the 33 storage cells has size $2log_2 n + 160 = 224$ bits, totalling 924 bytes of storage. If we output one hash value per second, such a chain would last for more than 68 years.

Our technique can be illustrated by the following example. Say that we can store three values (which we refer to as "pebbles") to represent the hash chain. Traditional techniques would place pebbles (i.e., store values) at the endpoint (whose position is n for a chain of length n), at $2n/3$, and at $n/3$. The computation needed to generate the next value would be between 0 and $n/3-1$. Instead, we put a pebble at n, at $n/2$, and at $n/4$. The first $n/4$ outputs would cost between 0 and $n/4-1$ to compute (given the pebble at $n/4$), and so would the next $n/4$ (given the pebble at the midpoint). Since the pebble at $n/4$ is useless once this point has been reached, we could instead move this pebble to position n, and for each available cycle (when we do not use all $n/4-1$ cycles), move it "downstream" towards the point $3n/4$. It will arrive there before we output the value at $n/2$. Thus, given that we now have a pebble at $3n/4$, and one at n, the remaining half of the traversal space will also have a maximum computational cost of $n/4-1$ per element. Given more pebbles, this can be further reduced.

Source code corresponding to the algorithm is available on the author's website [3], along with the full paper.

Acknowledgments

Many thanks to Ari Juels for help simplifying the algorithm, and to Gustav Hast, Adrian Perrig, Tal Rabin and Leo Reyzin for helpful suggestions and valuable discussions. Thanks to Helger Lipmaa for implementing the algorithm.

References

[1] "Desirable Dust", A survey about the real-time economy, The Economist, Feb 2 '02, pp. 8–9.

[2] G. Itkis and L. Reyzin, "Forward-Secure Signatures with Optimal Signing and Verifying," Crypto '01, pp. 332–354.

[3] M. Jakobsson, Full paper and implementation, www.markus-jakobsson.com

[4] S. Micali, "Efficient Certificate Revocation," Proceedings of RSA Conference 1997, and U.S. Patent No. 5,666,416.

[5] A. Perrig, R. Canetti, D. Song, and D. Tygar, "Efficient and Secure Source Authentication for Multicast," Proceedings of Network and Distributed System Security Symposium NDSS 2001, February 2001.

[6] A. Perrig, R. Canetti, D. Song, and D. Tygar, "TESLA: Multicast Source Authentication Transform", Proposed IRTF draft, http://paris.cs.berkeley.edu/ ~ perrig/

[7] K. S. J. Pister, J. M. Kahn and B. E. Boser, "Smart Dust: Wireless Networks of Millimeter-Scale Sensor Nodes. Highlight Article in 1999 Electronics Research Laboratory Research Summary.", 1999. See http://robotics.eecs.berkeley.edu/ ~ pister/SmartDust/

[8] R. Rivest and A. Shamir, "PayWord and MicroMint–Two Simple Micropayment Schemes," CryptoBytes, volume 2, number 1 (RSA Laboratories, Spring 1996), 7–11.

[9] FIPS PUB 180-1, "Secure Hash Standard, SHA-1," www.itl.nist.gov/fipspubs/fip180-1.htm

[10] S. Stubblebine and P. Syverson, "Fair On-line Auctions Without Special Trusted Parties," Financial Cryptography '01.

[11] "Where's the smart money?", Science and Technology, The Economist, Feb 9 '02, pp. 69–70.

[1] Intellectual rights to this work held by the author.