Skip Ratchet

A Hierarchical Hash System

Brooklyn Zelenka
Fission Codes
brooklyn@fission.codes

Abstract

Hash chains are a simple way to generate pseudorandom data, but are inefficient in situations that require long chains. This can cause unnecessary overhead for use cases including logical clocks, synchronizing the heads of a pseudorandom stream, or noninteractive key agreement. This paper presents the "skip ratchet", a novel pseudorandom function that can be efficiently incremented by arbitrary intervals.

1 Introduction

Hash algorithms provide a way to generate pseudorandom data from an input that is impractical to reverse. Many applications iteratively hash a value ("ratcheting" [4]) for pre-computation resistance, backward secrecy, one-time schemes (e.g. S/KEY [2]), deterministically generating unique names, key derivation functions, digital payment schemes [6], and so on.

This paper introduces the "skip ratchet", a hierarchical and iterative hashing system suitable as an extension to pseudorandom (PRF) and key derivation (KDF) functions. A skip ratchet is capable of efficiently making large leaps in hash count while preserving backward secrecy in a passive setting.

2 Motivation

The intersection of private data and location-independent access control in open protocols is only beginning to be explored. Decentralization, trust minimization, and non-interactive protocols are increasingly important, but have many unsolved problems including how to efficiently secure changing data in an unknown and unstable topology. Not knowing the number or attributes of peers implies the use of passive security methods based directly on data rather than mediated by live processes.

An increasing number of applications in open decentralized networks have no fixed topology, and require non-interactive key agreement on a large number of files, the ability to share a single file (and no others), a range of files, or elements from a point in time onwards (but not prior versions). As such, there is also a need to keep the historical information of the internal state secret from even those with current access.

One approach for securing history is to use a simple ratchet function, iteratively hashing on each update to produce the next state (a "hash chain"). This works well if the number of changes to synchronize is small, but $\mathcal{O}(n)$ ratchet steps is prohibitive as the difference grows. It allows a malicious participant to force others to a large amount of work in order to access the latest update.

The skip ratchet presented here improves the situation. A skip ratchet is able to synchronize in sublinear time, supports efficient arbitrary access in the forward direction, maintain secrecy of its internal state, and does not leak metadata such as the number of updates or participants. Further, jump size and granularity are configurable.

3 Numeral Intuition

The problem of how to efficiently calculate the nth step of a ratchet in fewer than n steps is structurally similar to how we efficiently represent large numbers. There are many systems for doing so. The two most common numeric representations are unary and positional numeral systems. The skip ratchet uses analogies to both of these concepts.

3.1 Unary Hashing

Hash chains are structurally very similar to unary counting systems. Unary is very concrete: the symbol count is equivalent to the number being represented. Two examples are tally marks and Peano numbers [8]. Replacing the successor function with a hash function h, one may represent unary counting with hashes.[1] As will become useful later, this does not make reference to a concrete "zero" value; merely an interval from the input x.

$$Peano(3) \Rightarrow S(S(S(0)))$$
$$h_{chain}(3) \Rightarrow h(h(h(I))) \equiv h^3(I)$$

Figure 1: Peano Number $\leftrightarrow$ Hash Chain

3.2 Positional Hashing

A major advantage of positional numerals is the ability to express large jumps with minimal effort. In positional systems (such as the familiar binary, decimal, and hexadecimal) each position corresponds to some factor for the numeral at that position.[2]

Just as positional numerals combine multiple numbers to represent a single sum, the hash count state

$$246_{16} = 2{\times}16^2 + 4{\times}16^1 + 6{\times}16^0 = 582_{10}$$

Figure 2: Componentized Hexadecimal

can have multiple components that are combined to create a single symmetric key.[3]

$$h_{10}(x + 0) \Rightarrow I$$
$$h_{10}(x + 11) \Rightarrow \langle h(J), h(I) \rangle$$
$$h_{10}(x + 582) \Rightarrow \langle h^5(K), h^8(J), h^2(I) \rangle$$

Figure 3: Compound Hash State

Unlike a positional number system used in arithmetic, a compound ratchet "forgets" its internal state, combining elements and flattening its structure. There are many methods of combination, including further hashing, concatenation followed by HKDF, and so on. This paper will use bitwise XOR (Figure 4).

$$h_{pos10,\oplus}(0) \Rightarrow I$$
$$h_{pos10,\oplus}(11) \Rightarrow h(J) \oplus h(I)$$
$$h_{pos10,\oplus}(582) \Rightarrow h^5(K) \oplus h^8(J) \oplus h^2(I)$$

Figure 4: Compound Hash Key Derivation

4 Skip Ratchet

The skip ratchet's state is built from a unary digit U, a fixed number n of positional digits, and their base b. The positional digits are bounded by the base, and so must track a natural number count. The unary digit explicitly does not track its count.[4]

---

[1]By convention, this paper will use I, J, and K as independent fixed initial values, x as an arbitrary variable, and h for hashing

[2]While most systems use fixed-base exponential factors, there exist systems where there is no fixed relationship between positions, or some other rule relating them.

[3]Different explorations of this core idea include hash calendars [1], and multidimensional hash chains (MDHC) [6]

[4]Note that Figure 5 is given as little-endian. The order of the state does not strictly matter, but it is convenient to associate the index of the state with the exponent for the base that the position represents.

$$S : \langle \langle Count_0, Value_0 \rangle, \langle Count_1, Value_1 \rangle \ldots \langle Count_{n-1}, Value_{n-1} \rangle, U \rangle_b$$

Figure 5: Skip Ratchet Counter State

Since the skip ratchet's internal state is composed of a fixed number of "digits", there is a fixed maximum interval by which the ratchet can jump. While it is possible to create variants that deterministically generate increasingly large digits as needed, this would leak data about the range of the counter. Indistinguishability is desirable in many use cases.

While advancing the ratchet by arbitrary intervals is possible, the jump operation is not the same as adding arbitrary integers. Incrementing all but the lowest digit cascades down to the lower values, "zeroing" them out. Therefore, arbitrary jumps require a carry (Algorithm 4). This situation is structurally similar to deterministic skip lists [5] (Figure 6), but with each step pointing to a monotonically increasing value, rather than previous ones.

Each digit is given a label and maximum value for all but the largest, which acts as a linear "spine". This largest element is analogous to an unbounded unary digit. The remaining elements are arranged positionally, and are bounded by their numerical base. Since the unary digit is a hard limit on the number of digits, it conceals information about the range of numbers that the current internal state represents. Any range within a fixed larger value is called an "epoch" (e.g. "the 300s epoch" or "the $h^{42}(x)$ epoch").[5]

4.1 Initialization

A skip ratchet is deterministically derived from a hash function h, a numeric base b, a positional digit count n, a function k to derive a key from the state, and an initialization vector IV. We distinguish between configurations by referring to their $\langle n, b, h, k \rangle$ triple, such as $\langle 3, 256, \mathsf{SHA256}, \mathsf{HKDF} \rangle$ skip ratchet.[6]

To prevent leaking the iteration count, all of the positional values in the seed state are immediately incremented by a random value. This randomized origin is treated as the ratchet's starting state. Given the backward secrecy constraint, only relative values may be used unless this initial value is known.

Each positional value is generated from the salt-concatenated $(x \| y)$ preimage's hash of its larger neighbour, in a recursive cascade starting with the unary digit (Algorithm 1). This protects the information needed to derive the current state, and thus prevents leaking all of the values in that range. The salt must be consistent across all calls.

The number of digits is configurable. These should be chosen to a balance jump control (the number of intervals that can be skipped by), and step-function performance. If the number of digits is large, increment on the unary element or higher digits will have require multiple hash operations in each zero cascade.

In practice, $\langle 2, 256, \mathsf{SHA256}, \oplus \rangle$ and $\langle 2, 256, \mathsf{BLAKE3}, \oplus \rangle$ skip ratchets have performed well. The maximum leap is 256^2, so the state $i + 256^3$ is (approximately) the same amount of work as $i + 256$. Given the small constant factor of hardware-accelerated hashing and the monotonicity of a ratchet, relatively large bases are often viable.

4.2 Basic Operations

Basic operations on the skip ratchet follow from the rules set out during initialization. It can be incremented sequentially, or leap to the next "zero" of any digit. Any interval can be efficiently found by combining increments and skips (Figure 8).

It is sometimes helpful to distinguish between a small and large skip, and so the terms "skip" and "leap" may be used for this purpose, despite them functioning identically but on different positions. The term "increment" (or "inc") is only used for the special case of the smallest movement (by one).

Being a ratchet, it can never be "unwound" (there

---

[5]The term "epoch" is used here in the generic or Ethereum[11] sense, and is a slightly different from some other ratchet constructions such as the Double Ratchet[9].

[6]This implies that a $\langle 0, b, h, k \rangle$ skip ratchet would express a standard hash chain.
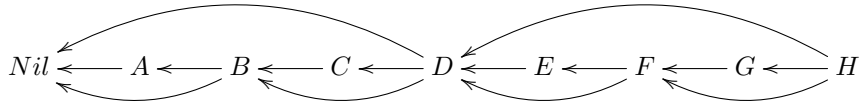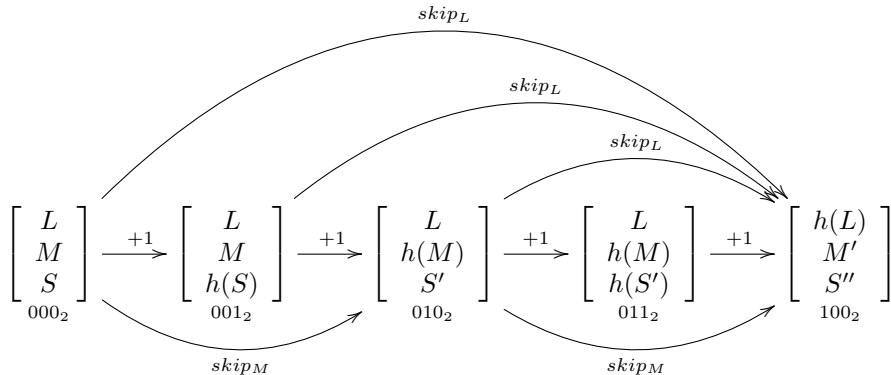
Figure 6: Deterministic Skip List

Figure 7: Simplified Skip Ratchet

is no inverse or subtractive analogue). Access to earlier keys requires knowing an even earlier state, and moving forward to the desired state from there. This is important for break-in resistance and backwards secrecy (e.g sharing a document from a point in time but no earlier).

4.2.1 Secret Derivation

Generating secrets or keys from the ratchet state can be done in a myriad of ways. This paper will use the bitwise XOR function, as it is straightforward and efficient.

4.2.2 Increment

As with positional numerals, when a digit reaches its maximum, the next larger digit is incremented, and all lower digits are zeroed out. The zero cascade works similarly for skip ratchets, but never repeats a value. Similar to initialization, each zero is derived from the preimage of its higher neighbour.

The special case of incrementing by one is the simplest way to demonstrate this (Algorithm 3). It functions by checking the lowest digit's counter, and if it's saturated, mark a carry and move to the next digit

and recurse, bounded by the unary digit. Any carried digits are then re-zeroed by the preimage of their larger neighbour.

4.2.3 Arbitrary Jumps

Jumps by an arbitrary interval require a little more calculation (Algorithm 4). The interval δ is componentized, and each digit incremented in ascending order. Every positional digit has an upper bound, so any remainder saved for the final step, where any zeroed digits are incremented to match the final count. Arbitrary jumps run in $\mathcal{O}(\log_b \delta)$ when $\delta < b^{b(n+1)}$ (the normal operating interval that the skip ratchet is tuned for). As the unary hash chain dominates as δ becomes very large, the complexity becomes $\mathcal{O}(\frac{\delta}{b^n})$.

5 Security

There is no inbuilt way to revoke this access once a secret is compromised. This is especially important if a skip ratchet is used to generate cryptographic keys. Updates rely on backward secrecy for protection, and key rotation in the case where a breach becomes known. Algorithms such as the Double Ratchet[9] have post-compromise security in a two-party semi-

Algorithm 1 Skip Ratchet Initialization

Require: $\langle n, b, \kappa \rangle \in \mathbb{N}_0 \times \mathbb{N}_1 \times \mathbb{N}_1$ ▷ Digit count, base, security parameter

1: $seed \overset{\$}{\leftarrow} \{0,1\}^\kappa$
2: $unary := hash(seed)$
3: $pos := []$
4: **for** $i \in n \ldots 0$ **do**   ▷ Descending to associate index with degree
5:      $\delta \overset{\$}{\leftarrow} 0 \ldots b - 1$ ▷ To increment by
6:      $seed := hash(salt\|seed)$ ▷ Secretly derive from the larger value
7:      $pos[i].count := \delta$
8:      $pos[i].value := hash^{\delta+1}(seed)$
9: **end for**
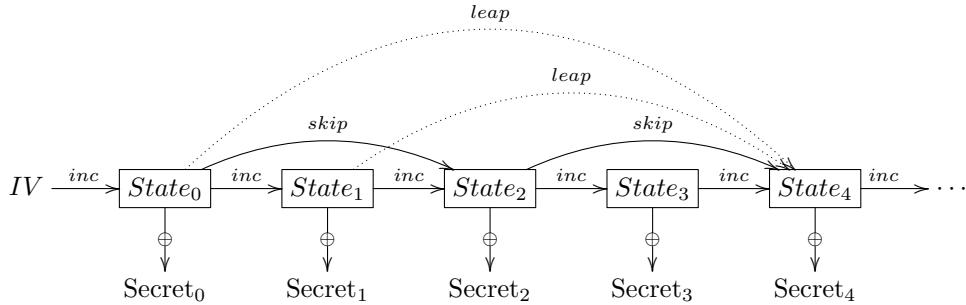10: **return** $unary, pos$

Figure 8: Binary Skip Ratchet Operations

trusted setup, where there key is automatically rotated as a matter of course.

The skip ratchet has two security domains: its external secret generation, and its internal state. The state is itself based on hierarchical hash chains to assure backwards secrecy, even for those that have been given access to its state. Not only is the previous state not calculable, other metadata such as the number of steps is also hidden, even when the internal state is known.

Revealing one derived key provides access to the cleartexts encrypted with that key (e.g. a single version of one file). Providing an XOR of the unary digit and a nonzero number of positional elements provides the remaining updates in that epoch. Access to the entire internal state grants provides keys from a point onwards, including potentially the initial state. The number of elements in the internal state must be at least $n + 2$ if intended for use in this setting, where n is the number of levels that are desired to be exposed.

5.1 Sharing Bounded State

The skip ratchet's internal state permits bounding derivation to a fixed range in a decentralized system. This is accomplished by only sharing some of the lower positions, with the rest being presented as a flattened hash.

For example, in a $\langle 3, 256, \mathsf{SHA256}, \oplus \rangle$ skip ratchet, sharing $\langle S, M, L \oplus U \rangle$ allows the recipient to move freely in the "medium" jump range, up to $i + 256^2 - 1$. Keys can still be fully generated thanks to XOR's associativity, but the recipient will need to ask for the next $L' \oplus U$ to continue.

Algorithm 2 Generating a Key

1: **function** TOKEY(ratchet)
2:     $sum := ratchet.unary$
3:     **for** $digit \in ratchet.pos$ **do**
4:         $sum := acc \oplus digit.value$
5:     **end for**
6:     **return** sum
7: **end function**

Algorithm 3 Incrementing a Skip Ratchet

1: **function** INC($ratchet$)
2:     $\{base, unary, pos\} \leftarrow ratchet$
3:     $seed := unary$
4:     $counter := length(pos)$
5:
6:     **for** $i \in 0 \ldots length(pos) - 1$ **do**
7:         $\{count, value\} \leftarrow pos[i]$
8:         **if** $count < base - 1$ **then**   ▷ Position not saturated
9:             $seed := value$
10:            $counter := i$
11:            $ratchet.pos[i].count := count + 1$
12:            $ratchet.pos[i].value := hash(value)$
13:            **break**
14:         **end if**
15:     **end for**
16:
17:     **if** $counter = length(pos)$ **then**   ▷ All positional values were saturated
18:         $ratchet.unary := hash(unary)$
19:     **end if**
20:
21:     **if** $counter > 0$ **then**
22:         **for** $j \in (counter - 1) \ldots 0$ **do**   ▷ N.B. Descending
23:            $seed' := pos[j].value$
24:            $ratchet[pos][j].count := 0$
25:            $ratchet[pos][j].value := hash(salt \| seed)$
26:            $seed := seed'$
27:         **end for**
28:     **end if**
29:
30:     $ratchet$
31: **end function**

Algorithm 4 Skip Ratchet Arbitrary Jump

```
 1: function JUMP(ratchet, amount)
 2:     {base, unary, pos} ← ratchet
 3:     remaining := amount
 4:     seed := unary
 5:     carry := []
 6:     n := length(pos)
 7:
 8:     for i ∈ 0 . . . n − 1 do
 9:         if remaining = 0 then
10:             break
11:         end if
12:
13:         componentᵢ := remaining  mod baseⁱ
14:         remaining := remaining − componentᵢ
15:         δᵢ := component/baseⁱ                                    ▷ No remainder because ascending in steps
16:
17:         {count, value} ← pos[i]
18:         headroom := base − count − 1
19:
20:         if δᵢ > headroom then
21:             carry[i] := steps − headroom
22:         else
23:             ratchet.pos[i].count := count + δᵢ
24:             if remaining = 0 then
25:                 seed := hash^{δᵢ−1}(value)
26:                 ratchet.pos[i].value := hash(seed)
27:             else
28:                 carry[i] := 0
29:             end if
30:         end if
31:     end for
32:
33:     if remaining > 0 then
34:         componentᵤ := remaining  mod baseⁿ
35:         δᵤ := componentᵤ/baseⁿ
36:         ratchet.unary := hash^{δᵤ}(unary)
37:     end if
38:
39:     for j ∈ (length(carry) − 1) . . . 0 do                        ▷ N.B. Descending
40:         seed := hash^{carry[j]−1}(salt‖seed)
41:         ratchet.pos[j].value := hash(seed)
42:     end for
43:
44:     return ratchet
45: end function
```

7

5.2 Post-Compromise Security

The skip ratchet trades off general post-compromise self-healing (of the unary digit) for permissionless key agreement between an unbounded and growable set of peers. If the secured communications are between exactly two parties, the Double Ratchet is typically a better choice due to its post-compromise security. Such a model could be adapted to use a skip ratchet for decentralized key agreement with periodic rotations of the unary digit. In essence, this involves distributing a freshly initialized skip ratchet to a list of trusted peers on every i steps [10], or any other method of rotating the unary digit.

5.3 Seed Trust

Giving a single party the power to decide the initial seed is not desirable in some situations. There are many options for generating this value from components provided by many parties. Some common options include Diffie-Hellman key exchange and its variants (e.g. X3DH) and key ceremonies.

6 Conclusion

This paper has presented the skip ratchet, an algorithm for decentralized secrets and keys on streams of encrypted data with unbounded and changing participants. Arbitrary forward search in the expected operating range can be performed in $\mathcal{O}(\log_b \delta)$ time within a configurable bound (and $\mathcal{O}(\frac{\delta}{b^n})$ outside it), backwards access is impossible without the earlier internal state, and total counts can never be inferred from the internal state.

7 Acknowledgements

References

[1] Ahto Buldas, Andres Kroonmaa, and Risto Laanoja. *Keyless Signatures' Infrastructure: How to Build Global Distributed Hash-Trees.* URL: https : / / guardtime . com / files / BuKL13.pdf. (accessed: 2022-01-03).

[2] Neil M. Haller. *The S/KEY One-Time Password System.* RFC 1760. Feb. 1995. DOI: 10 . 17487/RFC1760. URL: https://rfc-editor. org/rfc/rfc1760.txt.

[3] Philipp Krüger and Brooklyn Zelenka. *WebNative Spiral Ratchet.* Fission Codes. 2021. URL: https : / / github . com / fission - suite / webnative / blob / e1eed4e750c668a6f54d9be701dc17e286b9eff4/ src/fs/data/private/spiralratchet.ts.

[4] Adam Langley. *Pond README.* 2012. URL: https : / / github . com / agl / pond / commit / 7bb06244b9aa121d367a6d556867992d1481f0c8.

[5] J. Ian Munro, Thomas Papadakis, and Robert Sedgewick. "Deterministic Skip Lists". In: *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 1992, pp. 367–375. URL: `https://www.ic.unicamp.br/~celio/peer2peer/skip-net-graph/deterministic-skip-lists-munro.pdf`. (accessed: 2021-12-26).

[6] Quan Son Nguyen. "Multi-Dimensional Hash Chains and Application to Micropayment Schemes". In: *CoRR* abs/cs/0503060 (2005). arXiv: `cs/0503060`. URL: `http://arxiv.org/abs/cs/0503060`.

[7] Brendan O'Brien. *go-wnfs Spiral Ratchet*. Qri. 2021. URL: `https://github.com/qri-io/wnfs-go/blob/62915924c866f7106aecacccd14e57bd1b62850f/private/ratchet/ratchet.go`.

[8] *Peano axioms §Arithmatic*. Wikipedia. URL: `https://en.wikipedia.org/wiki/Peano_axioms#Arithmetic`. (accessed: 2021-12-26).

[9] Trevor Perrin and Moxie Marlinspike. *The Double Ratchet Algorithm*. Version Revision 1, 2016-11-20. Signal Foundation. Nov. 2016. URL: `https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf`.

[10] Matthew Weidner et al. *Key Agreement for Decentralized Secure Group Messaging with Strong Security Guarantees*. Cryptology ePrint Archive, Report 2020/1281. `https://ia.cr/2020/1281`. 2020.

[11] Gavin Wood. *Ethereum: A Secure Decentralized Generalized Transaction Ledger*. Version Berlin (fabef25). Ethereum Foundation. Dec. 2021. URL: `https://ethereum.github.io/yellowpaper/paper.pdf`. (accessed: 2022-01-03).