Skip Ratchet

A Recursive Hash System

Brooklyn Zelenka
Fission Codes
brooklyn@fission.codes

April 2, 2022
v1.0.0

Abstract

Hash chains are a simple way to generate pseudorandom key material. However, they are inefficient in many situations, including when used as logical clocks, or for synchronizing the heads of a forward secret stream. This paper presents the "skip ratchet", a novel symmetric key derivation 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" [5]) for pre-computation resistance, forward secrecy (FS), one-time schemes (e.g. S/KEY [3]), deterministically generating unique names, key derivation functions (KDFs), digital payment schemes [8], and so on. This paper introduces the "skip ratchet", a hierarchical and iterative hashing system capable of efficiently making large leaps in hash count, while preserving forward secrecy in a passive setting.

2 Motivation

The intersection of private data and location-independent access control in open distributed systems is only beginning to be explored. Trust minimized protocols are increasingly important, but admit to many unsolved problems, including how to 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.

An increasing number of applications in open networks have no fixed topology, and need to work in presence of high latency or network partitions. They require zero-interaction key agreement on many items, the ability to share a single item (and no others), a range of items, or elements from a point onwards. As such, there is also a need to keep the historical information of the internal state secret.

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, maintains backwards secrecy of its internal state, and does not leak metadata such as the number of updates or participants. Further, jump size and granularity is configurable.

3 Numeral Intuition

The problem of how to efficiently calculate the n^{th} step of a ratchet in less than n steps is structurally similar to how we efficiently represent large numbers. There are many systems for doing so. The two most common are tally marks (unary) and positional numeral systems. The Spiral Ratchet uses both of these concepts.

3.1 Unary Hashing

Unary is one of the simplest forms of expressing numbers. This system is very concrete: the number of symbols is equivalent to the number being represented. Two examples are tally marks and Peano numbers [10].

Replacing the successor function with a hash function h, one may represent unary counting in a structurally equivalent manner to Peano numbers. Importantly, this does not make reference to a concrete "zero" value; merely an interval from the input x[1]. It is common to denote n applications of a hash h as $h^n(x)$. While this method is very straightforward, it is neither space nor time efficient.

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

Figure 1: Peano Number ↔ Hash Chain

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

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]

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

Figure 2: Componentized Hexadecimal

Just as positional numerals combine multiple numbers to represent a single sum, a KDF state can have multiple components that are combined to create a single symmetric key.[3]

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

Figure 3: Compound Hash State

Unlike a positional number system used in arithmetic, a KDF "forgets" its internal state, combining elements and flattening its structure. There are many methods of combination, including further hashing; this paper will use bitwise XOR (Figure 4).

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

Figure 4: Compound Hash Key Derivation

4 Bounding

Unlike numeral systems useful for arithmetic, the skip ratchet's internal state is composed of a fixed number of "digits". This limits the jumps to fixed intervals,

[2] While most systems use fixed-base exponential factors, there exist systems where there is no common relationship between positions.

[3] Different directions of this core idea include hash calendars [2], and multidimensional hash chains (MDHC) [8]

with a maximum jump interval. While it is possible to create variants that deterministically generate increasingly large digits as needed, but this would leak data about the range of the counter. Being computationally indistinguishable is desirable in many use cases, such as for securing updatable documents in an public network.

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 is structurally similar to deterministic skip lists [7] (Figure 5), but with each step pointing to a monotonically increasing value, rather than previous ones.[4]

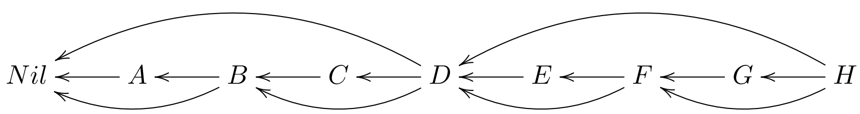

Figure 5: Deterministic Skip List

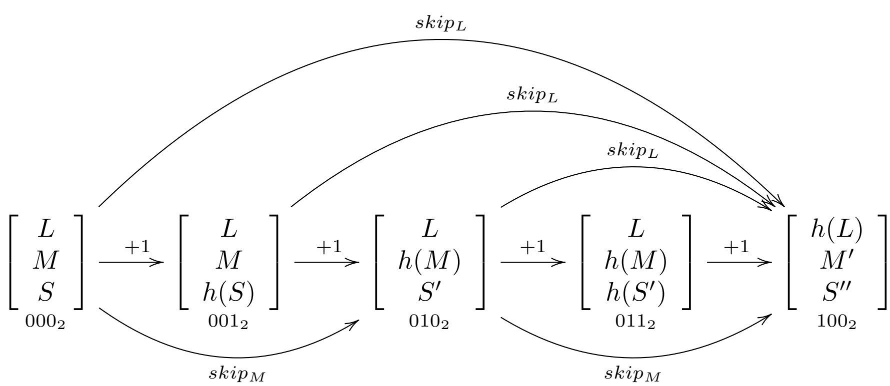

Figure 6: Simplified Skip Ratchet

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] While the analogy to fingered DAGs such as skip lists in the name is apt, the skip ratchet is distinct from other DAG-structured hash constructions like the Merkle Tree [6] or Hash Calendar [2]. A skip ratchet does not aggregate digests into a signature, but rather counts by iteratively hashing the same value repeatedly, in tiers.

[5] The term “epoch” is used here in the generic or Ethereum[13] sense, and is a slightly different from some other ratchet constructions such as the Double Ratchet[1].

By geometric analogy, the possible steps in this construction are similar to moving along a spiral: one component returns to its initial value, but the other dimension is always moving forward. Following a spiral in single steps is smooth, but leaps are possible by jumping to adjacent rings (Figure 7). Unlike the repeating numerals found in most counting systems, the skip ratchet never repeats a value despite the counter being kept to a limited number of elements. We call this the "spiral property.".

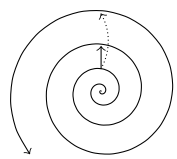

Figure 7: Spiral With Leaps

5 Skip Ratchet

5.1 Internal State

The Spiral 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.

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

Figure 8: Skip Ratchet State

Note that Figure 8 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.

5.2 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 the state, and an initialization vector IV. We distinguish between configurations by referring to their (n, b, h, k) triple, such as $(3, 256, \mathsf{SHA3}, \oplus)$ skip ratchet.

$$\langle \langle n, b, h, k \rangle, IV \rangle \to S$$

Figure 9: Seed Generation

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 *binary complement (∼) of the preimage* 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.

Algorithm 1 Skip Ratchet Initialization

Require: $\langle n, b, k \rangle \in \mathbb{N}_1 \times \mathbb{N}_1 \times \mathbb{N}_1$

$seed \xleftarrow{\$} \{0, 1\}^k$
$pos \coloneqq []$
$unary \coloneqq hash(seed)$
for $i \leftarrow (n-1) \ldots 0$ **do** ▷ Descending to associate index with degree
  $\delta \xleftarrow{\$} 0 \ldots b-1$
  $seed \coloneqq hash(\sim seed)$ ▷ Secretly derive from the larger value
  $pos[i].count \coloneqq \delta$
  $pos[i].value \coloneqq hash^{\delta+1}(seed)$
end for
return $b, unary, pos$

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, $(3, 256, \mathsf{SHA3}, \oplus)$ and $(3, 256, \mathsf{BLAKE3}, \oplus)$ 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.

5.3 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 10).

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 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).

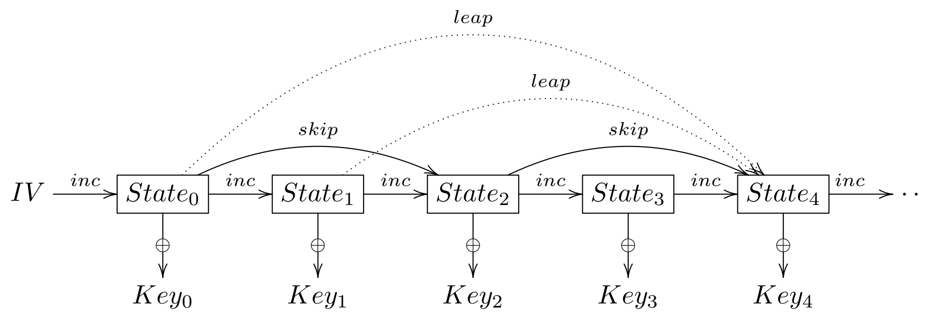

Figure 10: Binary Skip Ratchet Operations

5.3.1 Key Derivation

Generating output key material (OKM) 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.

Algorithm 2 Generating a Key

function TOKEY(ratchet)
    $sum := ratchet.unary$
    for $digit \in ratchet.pos$ **do**
        $sum := acc \oplus digit.value$
    end for
    return sum
end function

5.3.2 Incrementing

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 (i.e. the spiral property). As during initialization, the initialization, each zero is derived from the preimage of the higher digit.

The special case of incrementing by one is the simplest way to demonstrate this (Algorxithm 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.

5.3.3 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

Algorithm 3 Incrementing a Skip Ratchet

function $\textsc{inc}(ratchet)$
    $\{base, unary, pos\} \leftarrow ratchet$
    $seed \coloneqq unary$
    $counter \coloneqq length(pos)$

    for $i \leftarrow 0 \ldots length(pos) - 1$ **do**
        $\{count, value\} \leftarrow pos[i]$
        if $count \stackrel{?}{<} base - 1$ **then** ▷ Position not saturated
            $seed \coloneqq value$
            $counter \coloneqq i$
            $ratchet.pos[i].count \coloneqq count + 1$
            $ratchet.pos[i].value \coloneqq hash(value)\}$
            break
        end if
    end for

    if $counter \stackrel{?}{=} length(pos)$ **then** ▷ All positional values were saturated
        $ratchet.unary \coloneqq hash(unary)$
    end if

    if $counter \stackrel{?}{>} 0$ **then**
        for $j \leftarrow (counter - 1) \ldots 0$ **do** ▷ N.B. Descending
            $seed' \coloneqq pos[j].value$
            $ratchet[pos][j].count \coloneqq 0$
            $ratchet[pos][j].value \coloneqq hash(\sim seed)$
            $seed \coloneqq seed'$
        end for
    end if

    $ratchet$
end function

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 $\mathcal{O}(\frac{\delta}{b^n})$.

6 Security

Encryption is a form of direct access control. Much like how a bearer token may grant access to some resource, knowing a key grants access to the cleartext. There is no inbuilt way to revoke this access once the key is compromised. Updates rely on forward secrecy for protection, and key rotation in the case where a breach becomes known. Algorithms like the Double Ratchet[11] have post-compromise security, where there key is automatically rotated as a matter of course. This necessarily requires trusting the other party.

The skip ratchet has two security domains: its external key generation, and its internal state. The internal state is itself based on hierarchical hash chains to assure forward secrecy, even for those that have been given access to the KDF's internal 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.

6.1 Interval Bounding

The novel construction of the skip ratchet permits for bounding further derivation to a fixed range. This is accomplished by only sharing a few of the lower positions, and only sharing the rest of te state as a flattened hash.

Revealing one OKM provides access to the cleartexts encrypted with that key (e.g. a single version of one file). Providing the XOR of the unary digit and a nonzero number of positional elements gives (post-compromise) access to the remaining number of updates in that epoch. Access to the entire internal state grants access from a point onwards, including potentially the initial state.

For example, in a $(4, 256, \mathsf{SHA3}, \oplus)$ skip ratchet, sharing $\langle S, M, L \oplus U \rangle$ allows the recipient to move freely in the "medium" jump range, up to $i + 256^3 - 1$. 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.

6.2 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 over time. In essence, this involves distributing a freshly initialized

Algorithm 4 Skip Ratchet Arbitrary Jump

function $\textsc{jump}(\mathit{ratchet}, \mathit{amount})$
    $\{\mathit{base}, \mathit{unary}, \mathit{pos}\} \leftarrow \mathit{ratchet}$
    $\mathit{remaining} \coloneqq \mathit{amount}$
    $\mathit{seed} \coloneqq \mathit{unary}$
    $\mathit{carry} \coloneqq []$
    $n \coloneqq \mathit{length}(\mathit{pos})$

    for $i \leftarrow 0 \ldots n-1$ **do**
        if $\mathit{remaining} \stackrel{?}{=} 0$ **then**
            break
        end if

        $\mathit{component}_i \coloneqq \mathit{remaining} \mod \mathit{base}^i$
        $\mathit{remaining} \coloneqq \mathit{remaining} - \mathit{component}_i$
        $\delta_i \coloneqq \frac{\mathit{component}}{\mathit{base}^i}$ ▷ No remainder because ascending in steps

        $\{\mathit{count}, \mathit{value}\} \leftarrow \mathit{pos}[i]$
        $\mathit{headroom} \coloneqq \mathit{base} - \mathit{count} - 1$

        if $\delta_i \stackrel{?}{>} \mathit{headroom}$ **then**
            $\mathit{carry}[i] \coloneqq \mathit{steps} - \mathit{headroom}$
        else
            $\mathit{ratchet}.\mathit{pos}[i].\mathit{count} \coloneqq \mathit{count} + \delta_i$
            if $\mathit{remaining} \stackrel{?}{=} 0$ **then**
                $\mathit{seed} \coloneqq \mathit{hash}^{\delta_i - 1}(\mathit{value})$
                $\mathit{ratchet}.\mathit{pos}[i].\mathit{value} \coloneqq \mathit{hash}(\mathit{seed})$
            else
                $\mathit{carry}[i] \coloneqq 0$
            end if
        end if
    end for

    if $\mathit{remaining} \stackrel{?}{>} 0$ **then**
        $\mathit{component}_u \coloneqq \mathit{remaining} \mod \mathit{base}^n$
        $\delta_u \coloneqq \frac{\mathit{component}_u}{\mathit{base}^n}$
        $\mathit{ratchet}.\mathit{unary} \coloneqq \mathit{hash}^{\delta_u}(\mathit{unary})$
    end if

    for $j \leftarrow (\mathit{length}(\mathit{carry}) - 1) \ldots 0$ **do** ▷ N.B. Descending
        $\mathit{seed} \coloneqq \mathit{hash}^{\mathit{carry}[j]-1}(\sim \mathit{seed})$
        $\mathit{ratchet}.\mathit{pos}[j].\mathit{value} \coloneqq \mathit{hash}(\mathit{seed})$
    end for

    return $\mathit{ratchet}$
end function

skip ratchet to a list of trusted peers on every i steps [12], or any other method of rotating the unary digit.

7 Conclusion

This paper has presented the skip ratchet, an algorithm for decentralized key agreement 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, backwards access is impossible without the earlier internal state, and total counts can never be inferred from the internal state.

8 Acknowledgements

The skip ratchet algorithm was developed at Fission Codes to support work on the WebNative File System (WNFS). Many thanks to the Fission Codes team for providing the support and freedom to develop this algorithm.

Thanks to Philipp Krüger at Fission Codes for feedback on the design, for writing the bulk of the TypeScript implementation[4], and for his patience as various parts of the algorithm were tweaked while that code was already in-flight.

Many thanks to Quinn Wilton at Fission Codes for her feedback on various drafts of this paper, research into the broader literature, and suggestions on intended audience.

Thanks to Brendan O'Brien at Qri for the Go implementation[9], and for many discussions on the best way to explain the concept.

Thanks to Daniel Holmgren at Bluesky for his feedback on early versions of this algorithm.

Thanks to Juan Benet at Protocol Labs for encouraging me to write a fuller description of the skip ratchet than what was available in the WebNative online documentation.

References

[1] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. *The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol.* Feb. 2020. URL: `https://eprint.iacr.org/2018/1037.pdf`.

[2] 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).

[3] 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`.

[4] 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`.

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

[6] Ralph C. Merkle. "A Digital Signature Based on a Conventional Encryption Function". In: *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*. CRYPTO '87. Berlin, Heidelberg: Springer-Verlag, 1987, pp. 369–378. ISBN: 3540187960.

[7] 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).

[8] 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`.

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

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

[11] 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`.

[12] 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.

[13] 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).