Beyond the Event Horizon of Keccak

Extending the Observation from 2011 and applications

Danilo Gligoroski
Norwegian University of Science and Technology (NTNU)
danilog@ntnu.no

12.11.2026

Abstract

In 2011 we observed that a single path in the Keccak sponge from a state whose capacity has all zero bits back to such a state yields an explicit, infinite family of second preimages for every message of at least one full rate block. Finding such a path costs about $2^{c/2}$, which lies "beyond the event horizon" for SHA3-256 ($c = 512$). In this paper we give four novel theorems around that observation. First, the capacity can have any value x: a two-block meet-in-the-middle loop from x back to x, found at heuristic cost $\approx \sqrt{\pi} \cdot 2^{c/2}$, gives the same unlimited family of second preimages requiring no further search, with connectors to and from x at the same expected cost each (Theorem 1). Theorem 2 extends this to k-block loops and to cycles of the capacity map. A single-state bridge, a state S with $\mathrm{cap}\, f(S) = \mathrm{cap}\, f^{-1}(S)$, where cap denotes the capacity part of the state, also gives such a family but costs $\approx 2^c$. Theorem 3 gives cheap collisions and a bridge-amplified multicollision: one inner collision and one connector yield equal-length Joux-style 2^k-multicollisions for every k, instead of Joux's k collision searches. Given a prefix reaching x and k bridges on x with pairwise-distinct first rate inputs, Theorem 4 gives k^L distinct equal-length messages of $2L + 1$ blocks reaching one full state, for every L; in the balanced search model, finding the bridges costs about $\sqrt{4k} \cdot 2^{c/2}$ calls for large k, independently of L. A binary Joux construction producing at least as many messages needs about $\lceil L \log_2 k \rceil$ collision searches, each of generic order $2^{c/2}$. To our knowledge, the k^L count of equal-length messages from $k > 1$ bridges at an arbitrary x is new. We give a verified full-round Keccak-f[1600] bridge with $c = 80$ and a full-round $c = 80$ example from the standard IV: an IV transition and three bridges ($2^{42.61}$ calls in total) give $3^3 = 27$ distinct 1710-byte messages with a common digest.

At $c = 88$ we found four bridges on one generalized capacity; two are published, and the other two are posed as one-sided forgery challenges, in which only the forward half of a bridge is public, with heuristic generic completion cost about 2^{88} (and 2^{92} for one of them, read at its larger effective capacity). Lemma 5 shows that a bridge at capacity c is a bridge at every smaller capacity. The bridges were found by an incremental distinguished-point search that resumes from its stored table; the soundness of its output and the distinctness of the first halves of the bridges it reports are machine-checked in Lean 4.

The second part of the paper offers novel protocols built on the asymmetry between minting and forging Keccak bridges. They include proof of work with hardness-adaptive capacity, dual-capacity commitments anchored by the real SHA3-256 digest, bridges as zero-knowledge-committed tokens, and a self-enforcing bounty market for reduced-round security. To our knowledge, these are the first unkeyed Keccak-based protocols that deliberately set the capacity so that the generic bridge cost is a feasible, tunable hardness parameter. We give a list of 13 further protocol ideas for application of Keccak bridges.

We prove the four theorems, machine-check the stated algebraic results in Lean 4, and provide reproducible experiments and open-source proof-of-concept code. The applications are design proposals whose end-to-end security remains open.

Keywords: Keccak, SHA-3, sponge construction, second preimages, multicollisions, zero-knowledge tokens, Lean 4.

AI disclosure. *Substantial AI contribution*, in the sense of the IACR AI Policy [43]. I carried out this work in cooperation with AI assistants, Google Gemini in the early phase and then significantly with Grok Bot, which contributed to ideas, code, experiments and verification. Some of my ideas about extending the previous observation and turning the newly discovered properties into useful protocols were tested, purified and improved by Grok Bot. In turn, its feedback stimulated new ideas, so the protocols emerged from an iterative exchange. AI contributions are delineated in the Acknowledgments. We used OpenAI Astra for the final technical review and proofreading. I checked all results and take full responsibility for them; no AI system is an author of this paper.

Contents

1 Introduction **4**
1.1 Related work . . . 6

2 Bridges and new results for Keccak beyond event horizon **7**
2.1 Background and conventions . . . 7
2.2 The 2011 observation . . . 7
2.3 Bridges at an arbitrary state . . . 7
2.4 A worked example . . . 13
2.5 Consequences . . . 14
2.6 Search algorithms . . . 17
2.7 A public Keccak capacity-bridge challenge . . . 20
2.7.1 One-sided forgery challenges . . . 21

3 Bridge multicollisions: k^L for every length **22**
3.1 Setup . . . 22
3.2 Full-state multicollisions . . . 23
3.3 Related work and positioning . . . 25
3.4 Demonstrations . . . 25
3.4.1 A full standard-IV example at $c = 80$ . . . 25
3.4.2 Four bridges at a generalized capacity, $c = 88$ . . . 27
3.5 Downward closure and lucky upgrades . . . 28

4 Applications: Building Cryptographic Protocols with Keccak Bridges **30**
4.1 Proof-of-work with hardness-adaptive capacity . . . 31
4.2 Bridges as zero-knowledge tokens . . . 32
4.2.1 A cheaper variant: publishing only the forward rate half . . . 36
4.2.2 A zero-knowledge signature from split bridges . . . 38
4.2.3 Other potential applications of Keccak bridges . . . 39

5 Conclusion **41**

Acknowledgments **42**

A Artifact and formal proofs **42**

B Vectors of the $c = 80$ standard-IV 3^3-multicollision **44**

C Public halves of the one-sided challenges **46**

D Test vectors **46**

E Experimental results **47**
E.1 Literal bridge vs. birthday connector search, $c \in \{16, 20, 24\}$ . . . 47
E.2 Birthday connector search, $c = 32$ . . . 48
E.3 Capacity-map functional graphs . . . 48
E.4 Concrete instances on the full Keccak-$f[1600]$ with $c = 32$ . . . 50

1 Introduction

The sponge construction [12] with capacity c is proven to resist generic inner-collision and path-finding attacks up to about $2^{c/2}$ permutation calls [12, 52]. In 2010–2011, during the SHA-3 competition, we noted that once an adversary knows a non-empty path from inner state 0^c to inner state 0^c, she obtains infinitely many second preimages of any message of at least r bits [34]; the Keccak team discusses this observation in their sponge monograph ([12], §5.10.2).

The cost of such a path for SHA3-256 is $\approx 2 \cdot 2^{256} = 2^{257}$ calls, which already exceeds the generic 2^{256} second-preimage bound for a 256-bit digest. So the observation is a structural property, not an attack. The question that motivates this paper is what happens *beyond* that horizon: if one bridge is ever found (for a reduced number of rounds, a smaller capacity, or by a future algorithm), what does it buy, how should one search for it, and can its generic cost be put to constructive use?

Origin of this work. This work started after my visit to Radboud University Nijmegen on the occasion of the public PhD defence of Koustabh Ghosh (05.10.2026) [31], and my meeting there with Joan Daemen and Gilles Van Assche. Recollections of the NIST SHA-3 standardization competition, and of my earlier observation about properties of the Keccak sponge construction [34], inspired me to revisit that old work.

Non-claim. We stress that nothing in this paper breaks Keccak or SHA-3: experiments use reduced rounds and small widths, and the structural results do not beat the generic bounds of [12, 52] for full SHA3-256.

Contributions.

1. Definition of a *connector*, a block sequence that leads the sponge from one capacity value to another, or from one full state to another (a *two-point connector*; Definition 1), and of a *bridge*, a connector that returns the sponge to a given capacity value x from a given rate (Definition 2), with a repetition lemma (Lemma 1), the special case of a *single-state bridge* S with an elementary proven identity (Proposition 1), and its consequences: second preimages, and cheap collisions and bridge-amplified multicollisions (Theorem 3), which give equal-length Joux-style 2^k-multicollisions from one inner collision and one connector, at a search cost of $\approx 3 \cdot 2^{c/2}$ independent of k instead of k collision searches (a saving of a factor of about $k/2.4$ in search cost, without changing the exponent $c/2$; Section 2); and, from k bridges on the same capacity value x (using inner-state paths [12, §5.5]), k^L full-state multicollisions of equal length for every L at a fixed heuristic cost of $\Gamma(k+\frac{1}{2})/\Gamma(k) \cdot \sqrt{2 \cdot 2^c/q}$, $q = 2p(1-p)$ (about $\sqrt{4k} \cdot 2^{c/2}$ at $p = \frac{1}{2}$), independent of L (plus one IV transition of $\approx \sqrt{\pi} \cdot 2^{c/2}$ when $x \neq 0^c$), against about $\lceil L \log_2 k \rceil$ collision searches of generic order $2^{c/2}$ each for a binary Joux construction [46] with at least as many messages (Theorem 4, Section 3; a full-round $c = 80$ example from the standard IV with 27 messages is in Section 3.4.1, and four bridges on one generalized capacity at $c = 88$ are in Section 3.4.2). Fixed-cost multicollisions are known for CubeHash [3, §3], and are also described by Ferguson, Lucks and McKay [29, §1.3], but they rely on the all-zero fixed state and give messages of different lengths; to our knowledge, the k^L count from $k > 1$ bridges at an arbitrary x is new. It has no impact on SHA3-256.

2. Explicit statements of the 2011 observation beyond the capacity with all zero bits (for the loop, a conjugation of the 2011 bridge, cf. [12, §5.5, §5.10.2]): free second preimages from a single-state bridge at an arbitrary state S (Proposition 3, cost $\approx 2^c$), Theorem 1, a two-block meet-in-the-middle loop from an arbitrary capacity value x back to x (heuristic cost $\approx 2 \cdot 2^{c/2}$), which contains the 2011 bridge as the special case $x = 0^c$, and Theorem 2,

its extension to k-block loops and to cycles of the capacity map. The gap between $2^{c/2}$ and 2^c is the minting/forging asymmetry used by our applications.

3. Search algorithms: the literal 2^c loop for single-state bridges, a two-table birthday search for two-point connectors (Proposition 2) with expected $\frac{\sqrt{\pi}}{2}2^{c/2}$ steps, the fixed-capacity meet-in-the-middle search of Theorem 1 for bridges, and a distinguished-point (parallel collision search) variant for large c, implemented and tested on $f[200]$ (Section 2.6).

4. Downward closure (Lemma 5): a bridge at capacity c is a bridge at every smaller capacity, and, conversely, a bridge is a "lucky" bridge at a larger effective capacity with probability 2^{-k} for k extra bits (2^{-2k} on a single capacity value; Remark 12).

5. One-sided forgery challenges on the full Keccak-$f[1600]$ at $c = 88$ and $c = 92$, based on downward closure and on Protocol 4, with a public verifier that accepts every valid completion, salted commitments and an explicit trust model (Section 2.7.1). The challenge data (the capacity value x, the public halves and the three commitments) are published identically in this paper (Table 2, Appendix C) and in the folder `challenge/one_sided/` of the artifact [33], so the paper and the archive fix a single, timestamped set of challenges.

6. Initial proposals of cryptographic protocols applying Keccak bridges, which use the generic hardness of bridges (Section 4): hardness-adaptive proof of work (Protocol 1), bridge tokens and their inheritance (Protocols 2 and 3), keys with only the forward rate half public (Protocol 4), and a zero-knowledge signature built from such keys (Section 4.2.2). They are designs, not fully verified protocols; only the zero-knowledge ownership circuits of Protocols 2 and 4 are prototyped and measured. To our knowledge, they are the first unkeyed Keccak-based constructions that choose the capacity on purpose so that the generic $2^{c/2}$ bridge cost becomes a feasible and tunable hardness parameter, rather than a bound kept out of reach (see Section 1.1 for the comparison). Section 4.2.3 lists 13 further protocol ideas, among them sealed bids and timed release, threshold custody, ring and group signatures, dual-capacity commitments and bridge-rooted zero-knowledge signatures, as open problems for the community.

7. Experiments on Keccak-$f[200]$ for $R = 1..18$: head-to-head literal bridge vs. birthday connector search, birthday connectors at $c = 32$, fixed-capacity MITM bridges for $c \in \{16, 20, 24\}$, distinguished-point bridges up to $c = 48$, exhaustive 2^{32} functional graphs of the capacity map, and fixed-rate sweeps (Appendix E). All are consistent with generic behaviour for $R \geq 2$, except the known $R = 2$ inverse capacity map.

8. An artifact (Appendix A), archived on Zenodo [33] (this paper uses version 1.7): folders of open-source proof-of-concept code (C, Python, SageMath, Go and Rust) for the experiments and zero-knowledge prototypes, the public verifier and the data of the one-sided challenges (quick tests in `run_all_tests.sh`; the full campaigns take hours to days), and a separate folder of Lean 4 proofs that machine-check the core identities (Propositions 1 and 2), the soundness of the abstract literal search, the soundness of the checker for birthday hits as two-point connectors, and the block-word algebraic parts of Theorems 2 and 3 (loops, repetition, insertion and second preimages; cheap collisions and the length, injectivity and same-state properties of the multicollisions); the k^L construction of Theorem 4 (composition, injectivity, length, common state, common suffix and the count) is machine-checked as well, as is the two-call soundness of the incremental distinguished-point collector and the pairwise distinctness of its first halves (Lemma 2) in `IncrementalSearch.lean`; capacity-map cycles and padding are proved on paper only, and all cost statements are heuristic estimates, not proofs.

1.1 Related work

The sponge monograph [12] (§5.5, §5.10.2) gives the $\approx 2^{(c+4)/2}$ path-to-inner-state meet-in-the-middle and discusses the 2011 observation; Lefevre–Mennink give tight sponge preimage bounds [52]. Sun et al. observe that the generic attacks matching the sponge's (second) preimage bounds use f^{-1}, and show that a Davies–Meyer feed-forward during absorption (SPONGE-DM) raises generic preimage resistance to n bits and second-preimage resistance to $\min\{n, c - \log_2 \alpha\}$ bits [71]; our bridges are exactly such inverse-permutation attacks. Multicollisions follow Joux [46]; memoryless claw search follows van Oorschot–Wiener [73].

Keccak with smaller capacity or width. Unkeyed designs that lower the capacity or the number of rounds still choose c so that $2^{c/2}$ stays out of reach: KangarooTwelve and TurboSHAKE use Keccak-$p[1600, 12]$ with $c = 256$ [16, 74], the Crunchy Contest uses $c = 160$, so that a generic inner collision costs $\approx 2^{80}$ calls [10], and ProgPoW uses Keccak-$f[800]$ with $c = 224$ [24]. For scale, the Bitcoin network computed on average about $1.0 \cdot 10^{21} \approx 2^{69.8}$ double-SHA-256 hashes per second in the 30 days to 23.10.2026 [18]. If we assume, generously in favour of the attacker, that dedicated hardware evaluates one Keccak-$f[1600]$ call as fast as one such hash, then 2^{80} calls take about 20 minutes of this global capacity, and one year of it is about $2^{94.7}$ calls. This is an order-of-magnitude estimate: no such Keccak hardware exists at that scale, and redirecting it would require retooling. Even so, under this hypothetical throughput assumption 2^{80} would be within reach of an actor of that size and not only through cryptanalysis, while 2^{128} stays far out of reach. Keyed designs with small width or capacity, such as Ketje [14], Keyak and its Motorist mode [15], and Farfalle/Kravatte [11], rely on keyed-duplex and keyed-sponge bounds [13, 2, 25, 47], which allow security beyond $2^{c/2}$ because the inner state depends on a secret key. Our bridges do not apply to them directly, since they would first need state recovery. Lightweight non-Keccak sponges such as PHOTON [39] and Quark [4] accept $c/2$-bit generic levels by design. None of these treats c as a deliberately feasible hardness parameter, which is how we use it in Section 4.

Preimages of round-reduced Keccak. Bernstein traded very large memory for a few bits of speed-up in second preimages for 6–8 rounds [9], and rotational cryptanalysis gave 4-round preimages of Keccak[r=1024, c=576] truncated to 512 bits [61]. Linear structures [38] linearise up to 3 rounds; they give 3-round SHAKE128 preimages with complexity 1 and the first practical 3-round Crunchy Contest solutions, and cross-linear structures [54] attack 3-round Keccak[r=240, c=160] and 3-round Keccak-256. Two-block preimages via the allocating approach [53] and its refinements [56, 40] make 3-round Keccak-224 practical, and Lin–He–Yu give an actual 3-round Keccak-256 preimage at about $2^{57.2}$ [57]. Non-linear structures [67], algebraic attacks [59], the polynomial method [26] and meet-in-the-middle [66] target the high-capacity Keccak-384/512. Internal differentials reach 5 rounds: Zhang–Hou–Liu attack 4-round SHA3-256 with $2^{151.5}$ Keccak calls and 5-round SHA3-256 with $2^{254.33}$ bit operations [78], and Lin et al. attack 5-round Keccak-256 with complexity 2^{230} [55]. On the c=160 Crunchy instances, practical preimages now reach 5 rounds for Keccak[r=640, c=160] [58, 10].

Collisions of round-reduced Keccak. Dinur–Dunkelman–Shamir combine a 1-round connector (target difference algorithm) with 3-round trails to find practical 4-round Keccak-224/256 collisions [27]; generalized internal differentials then give actual 3-round Keccak-384/512 collisions and attacks on 4-round Keccak-384 and 5-round Keccak-256 [28]. Linearising the first one or two rounds yields 2- and 3-round differential connectors [65, 70], which give practical 5-round SHAKE128 and Keccak-224 collisions and a 6-round Crunchy collision; the journal version [36] adds actual 5-round SHA3-224 and SHA3-256 collisions, still the practical record for SHA3-256. SAT-based search gives a 6-round SHAKE128 collision with $2^{123.5}$ time, but 6-round SHA3-224/256 collisions only in the quantum setting [37]. Two-block messages with a SAT connector give practical 4-round SHA3-384 collisions [41], and conditional internal differentials and probabilistic linearisation extend collisions to 4-round SHA3-512, 5-round SHA3-384 and

6-round SHAKE256 [76, 77]. A 2026 preprint reports a practical 6-round SHAKE128 collision with a 160-bit digest and 6-round near-collisions on SHA3-256 [72].

2 Bridges and new results for Keccak beyond event horizon

2.1 Background and conventions

Keccak-$f[b]$. The state has $b = 25w$ bits, $w \in \{8, 16, 32, 64\}$, so $b \in \{200, 400, 800, 1600\}$, arranged as 5×5 lanes of w bits. A round is $\iota \circ \chi \circ \pi \circ \rho \circ \theta$; the full permutation has $12 + 2\log_2 w$ rounds (24 for $b = 1600$, 18 for $b = 200$) [62]. χ is the only non-linear step and has degree 2; χ^{-1} has degree 3, and θ^{-1} is linear but dense. We do not consider the small widths $b = 25, 50, 100$; our experiments use $b = 200$ (and $b = 400$ for the algebraic $R = 2$ runs) up to $b = 1600$.

Sponge. With rate r and capacity c, $r + c = b$, a block $m \in \{0,1\}^r$ is absorbed as $T \mapsto f(T \oplus (m\|0^c))$, starting from $T = 0^b$. *Keccak-256* and *SHA3-256* both use $r = 1088$, $c = 512$; they differ only in padding: Keccak-256 pads with `0x01...0x80` (pad10*1), SHA3-256 appends the domain bits `01` first, giving `0x06...0x80`. Bridge blocks are full rate blocks before padding, so all constructions below are valid for both paddings, provided the first block of the original message carries no padding or suffix bits, i.e. $|M| \geq r$.

Our conventions. f_R denotes Keccak-$f[b]$ reduced to its *first* R rounds (round indices $0..R-1$ with their ι constants), not the last R rounds as in the FIPS 202 Keccak-p definition. For a state S, rate(S) is the first r bits and cap(S) the last c bits (a bit-level split, any c). All experiments reported here use the first R rounds. The accompanying source code (code release) also supports the FIPS 202 convention of the last R rounds (option `--last-rounds`) and an arbitrary start round (`--offset` K), with the matching ι constants.

2.2 The 2011 observation

Find $P_1, P_3 \in \{0,1\}^r$ with

$$f_R(P_1\|0^c) = (P_2\|C), \qquad f_R^{-1}(P_3\|0^c) = (P_4\|C).$$

Then from the zero state, absorbing P_1 and $P_2 \oplus P_4$ gives $f(P_4\|C) = (P_3\|0^c)$, and XOR-ing P_3 into the next block returns the full state to 0^b. Hence $P_1\|(P_2 \oplus P_4)\|P_3$ is a second preimage of 0^r, and

$$P_1\|(P_2 \oplus P_4)\|\big((P_3 \oplus P_1)\|(P_2 \oplus P_4)\big)^i\|(P_3 \oplus M_0)\|M_1 \cdots M_k$$

is a second preimage of any message M of at least r bits with padded blocks $M = M_0 \cdots M_k$ (so that M_0 contains no padding), for every $i \geq 0$ [34]. The generic cost is a birthday match on c bits, $\approx 2 \cdot 2^{c/2}$.

2.3 Bridges at an arbitrary state

We work with padded block sequences. Absorbing a block $m \in \{0,1\}^r$ into a state T gives $f(T \oplus (m\|0^c))$, and for a block sequence $B = (m_1, \ldots, m_j)$ we write $T \xrightarrow{B} T'$ for the state T' reached from T after absorbing $m_1, \ldots, m_j$ in order.

Definition 1 (Connector[1])**.** Let $u, v \in \{0,1\}^c$. A *connector from capacity u to capacity v* is a block sequence C with $(y\|u) \xrightarrow{C} (y'\|v)$ for some $y, y' \in \{0,1\}^r$. For states $S, S' \in \{0,1\}^b$,

[1]Not to be confused with the differential connectors of [65, 70], which act inside the first rounds of one call of f and link message pairs to the input difference of a differential trail; our connectors are value-level block sequences under the full permutation.

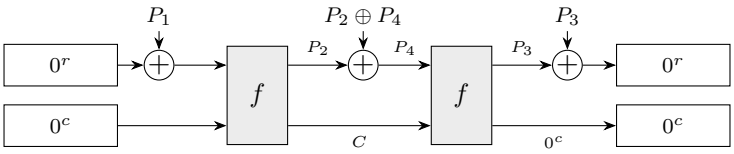

Figure 1: The three-block bridge of [34, Fig. 1], reproduced. With $f(P_1\|0^c) = (P_2\|C)$ and $f^{-1}(P_3\|0^c) = (P_4\|C)$, the two blocks $P_1 \,\|\, (P_2 \oplus P_4)$ lead from the zero state to $(P_3\|0^c)$, and XOR-ing P_3 into the next block resets the input of the next f call to 0^b; hence $P_1\|(P_2 \oplus P_4)\|P_3$ and the single block 0^r reach the same state.

a *two-point connector from S to S'* is a block sequence C with $S \xrightarrow{C} S'$; it is in particular a connector from $\operatorname{cap} S$ to $\operatorname{cap} S'$.

A bridge (Definition 2) is thus a connector from a capacity x to itself.

Definition 2 (Bridge)**.** Let f be a permutation of $\{0,1\}^b$, $b = r + c$, let $x \in \{0,1\}^c$, $y \in \{0,1\}^r$ and $k \geq 1$. A *k-block bridge at capacity x from rate y* is a block sequence $B = (m_1, \ldots, m_k)$ with

$$(y\|x) \xrightarrow{B} (y'\|x) \qquad \text{for some } y' \in \{0,1\}^r.$$

Its *rate offset* is $\delta = y \oplus y'$. The bridge is *closed* if $\delta = 0^r$, i.e. if $(y\|x) \xrightarrow{B} (y\|x)$.

Lemma 1 (Repetition)**.** *Let $B = (m_1, \ldots, m_k)$ be a k-block bridge at capacity x from rate y, ending in $(y'\|x)$, with rate offset $\delta = y \oplus y'$. Then $B' = (m_1 \oplus \delta, m_2, \ldots, m_k)$ is a closed k-block bridge at capacity x from rate y'. Hence, for every $i \geq 0$, $B \,\|\, (B')^i$ leads from $(y\|x)$ to $(y'\|x)$.*

Proof. From $(y'\|x)$, absorbing $m_1 \oplus \delta$ gives $f\big((y' \oplus m_1 \oplus y \oplus y')\|x\big) = f\big((y \oplus m_1)\|x\big)$, which is the state reached by absorbing m_1 from $(y\|x)$. The blocks $m_2, \ldots, m_k$ are the same as in B, so B' ends in the same state as B, namely $(y'\|x)$. Thus $(y'\|x) \xrightarrow{B'} (y'\|x)$, and B' is closed. The second claim follows by induction on i: the case $i = 0$ is the hypothesis on B, and if $B\|(B')^i$ leads to $(y'\|x)$, then one more copy of B' returns to $(y'\|x)$. □

The sequences $B\|(B')^i$, $i \geq 0$, are pairwise distinct, since they have the pairwise different lengths $k(i+1)$.

Definition 3 (Single-state bridge)**.** A state S is a *single-state bridge* if $\operatorname{cap} f(S) = \operatorname{cap} f^{-1}(S)$, i.e. $f(S) = (P_1\|x)$ and $f^{-1}(S) = (P_3\|x)$ for some $x \in \{0,1\}^c$ and $P_1, P_3 \in \{0,1\}^r$. Its *loop word* is $\Delta = P_1 \oplus P_3$.

Proposition 1 (Bridge identity)**.** *Let $S = (s\|z)$, $s \in \{0,1\}^r$, $z \in \{0,1\}^c$, be a single-state bridge with loop word Δ. Then $f(S) \oplus (\Delta\|0^c) = f^{-1}(S)$. Consequently, $(0^r, \Delta)$ is a closed 2-block bridge at capacity $z = \operatorname{cap}(S)$ from rate $s = \operatorname{rate}(S)$; that is, the two-block message $(0^r, \Delta)$ absorbed from state S returns to S.*

Proof. $f(S) \oplus (\Delta\|0^c) = (P_1 \oplus P_1 \oplus P_3 \,\|\, x) = (P_3\|x) = f^{-1}(S)$. With $y = s$ and capacity z, the starting state of Definition 2 is $(s\|z) = S$. Absorbing 0^r from S gives $f(S \oplus 0^b) = f(S)$; absorbing Δ next gives $f(f(S) \oplus (\Delta\|0^c)) = f(f^{-1}(S)) = S$. So $S \xrightarrow{(0^r,\Delta)} S$, the capacity z and the rate s are both restored, and the rate offset is 0^r. □

Note that $x = \operatorname{cap} f(S)$ and $z = \operatorname{cap} S$ are in general different: the bridge of Proposition 1 lives at the capacity z of S itself, while x is the capacity of the intermediate state $f(S)$.

Proposition 2 (Two-point version)**.** *If* $\operatorname{cap} f(S_i) = \operatorname{cap} f^{-1}(S_j) = C$*, with* $f(S_i) = (P_1 \| C)$*,* $f^{-1}(S_j) = (P_3 \| C)$*, then* $S_j = f\big(f(S_i) \oplus (\Delta \| 0^c)\big)$ *with* $\Delta = P_1 \oplus P_3$*; i.e. the message* $(0^r, \Delta)$ *leads the sponge from* S_i *to* S_j*, so it is a two-point connector from* S_i *to* S_j *(Definition 1). If moreover* $\operatorname{cap} S_i = \operatorname{cap} S_j$*, then* $(0^r, \Delta)$ *is a 2-block bridge at capacity* $\operatorname{cap} S_i$ *from rate* $\operatorname{rate} S_i$ *with rate offset* $\operatorname{rate} S_i \oplus \operatorname{rate} S_j$*.*

Proof. As in Proposition 1 with S_i, S_j in place of S. □

Remark 1 (Special cases of the bridge definition)**.**

1. *The 2011 bridge.* With the notation of Fig. 1, $0^b \xrightarrow{P_1} (P_2 \| C) \xrightarrow{P_2 \oplus P_4} f(P_4 \| C) = (P_3 \| 0^c)$. So $(P_1, P_2 \oplus P_4)$ is a 2-block bridge at the capacity with all zero bits from rate 0^r, with rate offset P_3. By Lemma 1, $(P_1 \oplus P_3, P_2 \oplus P_4)$ is a closed 2-block bridge at the capacity with all zero bits from rate P_3; these are exactly the repeated blocks of the family in Section 2. The third block P_3 of Fig. 1 cancels the rate P_3 before the next permutation call: in the absorption convention above, the three-block sequence $(P_1, P_2 \oplus P_4, P_3)$ leads from 0^b to $f(0^b)$, the same state as the single block 0^r, so it is used as the prefix of a following block $(P_3 \oplus M_0)$ rather than as a bridge on its own.

2. *The MITM loop.* In the notation of Theorem 1, $L(y) = (y \oplus a, u \oplus w)$ is a 2-block bridge at the arbitrary capacity x from rate y, ending in $(w' \| x)$, with rate offset $y \oplus w'$. The closed bridge B' given by Lemma 1 is $(y \oplus a \oplus y \oplus w', u \oplus w) = (w' \oplus a, u \oplus w) = L(w')$.

3. *Single-state bridges.* By Proposition 1, a single-state bridge S gives the closed 2-block bridge $(0^r, \Delta)$ at capacity $\operatorname{cap}(S)$ from rate $\operatorname{rate}(S)$.

4. *Cycles of the capacity map* (Theorem 2(ii)). Let $h(z) = \operatorname{cap} f(0^r \| z)$ and suppose $z_0 \mapsto z_1 \mapsto \cdots \mapsto z_\ell = z_0$ is a cycle of length ℓ of h. From any state $(y \| z_0)$, absorbing the current rate as the next block gives $f(0^r \| z_t)$, whose capacity is z_{t+1}; after ℓ such blocks the capacity is z_0 again. So for every y there is an ℓ-block bridge at capacity z_0 from rate y (the blocks depend on y), and likewise at each z_t on the cycle. For the inverse map $h'(z) = \operatorname{cap} f^{-1}(0^r \| z)$ one traverses the cycle backwards: if $h'(z_t) = z_{t+1}$ and $f^{-1}(0^r \| z_t) = (q_t \| z_{t+1})$, then from $(y \| z_{t+1})$ the block $y \oplus q_t$ leads to $(0^r \| z_t)$, so a cycle of h' of length ℓ gives an ℓ-block bridge at each capacity value on it, ending at rate 0^r, and closed when started from rate 0^r. Entering the cycle from the IV is a separate (generic) cost.

Remark 2. The 2011 bridge lives at the capacity with all zero bits ($P_1 \| 0^c$ forward, $P_3 \| 0^c$ backward), which is what allows it to be prepended to any message. For a single-state bridge S, the loop of Proposition 1 is usable inside any message whose absorption passes through S; reaching S from the IV costs a separate path search. Proposition 3 and Theorem 1 below show that this path search is all that is needed: a single-state bridge at *any* state, or a bridge at *any* capacity, gives the same free second preimages as the 2011 bridge. Formally, the loop at capacity x is the 2011 bridge for the conjugate permutation $\tau_x \circ f \circ \tau_x$ with $\tau_x(s) = s \oplus (0^r \| x)$, combined with the path-to-inner-state search of [12, §5.5]; we make it explicit because the protocols of Section 4 need loops at capacities other than 0^c. For second preimages alone it is not cheaper than the 2011 bridge ($\approx 3\sqrt{\pi} \cdot 2^{c/2}$ for one loop and two separately run connector searches, versus $\approx \sqrt{\pi} \cdot 2^{c/2}$).

Proposition 3 (Free second preimages from a single-state bridge at an arbitrary state)**.** *Let* S *be a single-state bridge with* $f(S) = (P_1 \| x)$*,* $f^{-1}(S) = (P_3 \| x)$ *and loop word* $\Delta = P_1 \oplus P_3$*. Let* $C_0 = (a_1, \ldots, a_s)$ *be any block sequence with* $0^b \xrightarrow{C_0} (y_0 \| x)$*, and let* $C_1 = (d_1, \ldots, d_t)$ *be any block sequence with* $S \xrightarrow{C_1} (y_1 \| 0^c)$*. Let* M *be any message of at least* r *bits, and let* $M_0 M_1 \cdots M_k$

($k \geq 1$) be its padded blocks, so that M_0 consists of message bits only. For every $i \geq 0$, the padded sequence

$$\widetilde{M}_i \;=\; C_0 \,\|\, (y_0 \oplus P_3) \,\|\, (0^r \| \Delta)^i \,\|\, C_1 \,\|\, (M_0 \oplus y_1) \,\|\, M_1 \cdots M_k$$

leads the sponge from 0^b to the same state as M, so $\widetilde{M}_i$ and M have the same digest for every output length. The messages $\widetilde{M}_i$, $i \geq 0$, are pairwise distinct.

Proof. By the choice of C_0, after absorbing C_0 the state is $(y_0 \| x)$. Absorbing $y_0 \oplus P_3$ gives $f\big((y_0 \oplus y_0 \oplus P_3) \| x\big) = f(P_3 \| x) = f(f^{-1}(S)) = S$. By Proposition 1, $(0^r, \Delta)$ is a closed bridge at capacity $\mathrm{cap}(S)$ from rate $\mathrm{rate}(S)$, so each copy absorbed from S returns to S, so after the i copies the state is still S (for $i = 0$ trivially). By the choice of C_1, absorbing C_1 from S gives $(y_1 \| 0^c)$. Absorbing $M_0 \oplus y_1$ then gives $f\big((y_1 \oplus M_0 \oplus y_1) \| 0^c\big) = f(M_0 \| 0^c)$, which is exactly the state after absorbing M_0 from 0^b. The remaining blocks $M_1, \ldots, M_k$ are identical in both sequences, so the final states, and hence all squeezed outputs, coincide. Since M_0 consists of message bits only, $\widetilde{M}_i$ is the padding of the message obtained from M by replacing its first r bits with $C_0 \| (y_0 \oplus P_3) \| (0^r \| \Delta)^i \| C_1 \| (M_0 \oplus y_1)$; its length differs from $|M|$ by a multiple of r, so its padding is the same as that of M, and $\widetilde{M}_i$ is itself a validly padded message. (For $|M| < r$ this can fail: for the empty message the modified block is in general not a valid padding.) Distinctness holds because the $\widetilde{M}_i$ have pairwise different lengths. □

Remark 3 (Cost and special cases)**.** The proposition is a construction; only the cost of its ingredients is heuristic. For an ideal permutation, a connector from capacity u to capacity v can be found by a meet-in-the-middle search on the c-bit capacity, run as a balanced complete-table (or distinguished-point, $p = \frac{1}{2}$) claw search of the same kind as in Theorem 4: the expected occurrence cost is $E[T_1] = \sqrt{\pi} \cdot 2^{c/2}$ calls of f and f^{-1}. This gives C_0 and C_1 at that expected cost each, when the two searches are run separately. Finding the single-state bridge S itself is different: the condition $\mathrm{cap}\, f(S) = \mathrm{cap}\, f^{-1}(S)$ is a c-bit condition on a *single* state, with no birthday pairing, so a generic search costs about 2^c calls of f and f^{-1} (our full-round f[200], $c = 32$ example took $2^{31.1}$ steps). The birthday-cost alternative is a two-block MITM bridge at capacity x for any x (Theorem 1; for $x = 0^c$ this is the 2011 bridge, Remark 1), which yields the same kind of loop at the same expected cost $\sqrt{\pi} \cdot 2^{c/2}$. Once found, (C_0, S, C_1) serves every message M of at least r bits, including messages chosen later. Two special cases: if $x = 0^c$, then C_0 is empty with $y_0 = 0^r$; and if the absorption of a given message passes through capacity x, the loop can be inserted there directly, with no C_0 or C_1 (Remark above). The 2011 bridge of Fig. 1 is the analogue for the capacity with all zero bits, which needs no connectors at all.

Theorem 1 (Birthday-cost loop at an arbitrary capacity)**.** *Let $x \in \{0,1\}^c$ and let $a, w' \in \{0,1\}^r$ satisfy the capacity match*

$$\mathrm{cap}\, f(a \| x) \;=\; \mathrm{cap}\, f^{-1}(w' \| x). \tag{1}$$

Write $f(a\|x) = (u\|v)$ and $f^{-1}(w'\|x) = (w\|v)$. Then from any state $(y\|x)$ the two blocks

$$L(y) \;=\; \big(y \oplus a,\; u \oplus w\big)$$

lead to the state $(w'\|x)$; that is, $L(y)$ is a 2-block bridge at capacity x from rate y with rate offset $y \oplus w'$, and $L(w')$ is a closed 2-block bridge at capacity x from rate w'. Hence, with C_0, y_0 as in Proposition 3 and C_1 a block sequence with $(w'\|x) \xrightarrow{C_1} (y_1\|0^c)$, the padded messages

$$C_0 \,\|\, L(y_0) \,\|\, L(w')^{\,i} \,\|\, C_1 \,\|\, (M_0 \oplus y_1) \,\|\, M_1 \cdots M_k, \qquad i \geq 0,$$

all have the same digest as $M = M_0 \cdots M_k$ for every message M of at least r bits, and are pairwise distinct.

Proof. From $(y\|x)$, absorbing $y \oplus a$ gives $f(a\|x) = (u\|v)$. Absorbing $u \oplus w$ gives $f(w\|v) = f(f^{-1}(w'\|x)) = (w'\|x)$. So $L(y)$ maps $(y\|x)$ to $(w'\|x)$, and $L(w')$ maps $(w'\|x)$ to itself (this is also Lemma 1 applied to $L(y)$). The rest is the argument of Proposition 3: after C_1 the state is $(y_1\|0^c)$, the block $M_0 \oplus y_1$ gives $f(M_0\|0^c)$, the remaining blocks are unchanged, and the messages have pairwise different lengths. □

Remark 4 (Cost of Theorem 1; many blocks)**.** Condition (1) is a c-bit match between two *independent* lists. Building a list of about $2^{c/2}$ images $\operatorname{cap} f(a\|x)$ and a list of about $2^{c/2}$ images $\operatorname{cap} f^{-1}(w'\|x)$ makes the expected number of matches equal to one; under the balanced complete-table model used in Theorem 4 (and for distinguished points with $p = \frac{1}{2}$), the expected occurrence cost of the search is $E[T_1] = \sqrt{\pi} \cdot 2^{c/2}$ calls of f and f^{-1}, rather than $2 \cdot 2^{c/2}$. Memoryless versions with distinguished points [73] have the same order. With two separately run connector searches C_0 and C_1, each of expected cost $\sqrt{\pi} \cdot 2^{c/2}$, the whole family has total expected occurrence cost $3\sqrt{\pi} \cdot 2^{c/2}$, against about 2^c for a single-state bridge. For $x = 0^c$ and $y_0 = 0^r$ the loop $L(0^r)$ is the 2011 bridge of Fig. 1: $a = P_1$, $u \oplus w = P_2 \oplus P_4$, $w' = P_3$ (Remark 1). Longer loops and cycles of the capacity map (Theorem 2 below) do not go below $2^{c/2}$ either. This is the generic $2^{c/2}$ inner-collision bound of the sponge [12], so the cost in Theorem 1 is optimal up to constants for an ideal permutation.

Theorem 2 (Loops of k blocks and cycles of the capacity map)**.** *Let f be a permutation of $\{0,1\}^b$, $b = r + c$, and $x \in \{0,1\}^c$.*

(i) k-block loops. *Let $k_1, k_2 \geq 1$, $a, \alpha_2, \ldots, \alpha_{k_1} \in \{0,1\}^r$ and $w', \beta_2, \ldots, \beta_{k_2} \in \{0,1\}^r$. Put $T_1 = f(a\|x)$, $T_t = f\big(T_{t-1} \oplus (\alpha_t\|0^c)\big)$ for $2 \leq t \leq k_1$, and $(u\|v) = T_{k_1}$; put $Z_{k_2} = (w'\|x)$, $Z_{t-1} = f^{-1}(Z_t) \oplus (\beta_t\|0^c)$ for $k_2 \geq t \geq 2$, and $(w\|v') = f^{-1}(Z_1)$. If $v' = v$, then for every $y \in \{0,1\}^r$ the $(k_1 + k_2)$-block word*

$$L(y) = \big(y \oplus a,\ \alpha_2, \ldots, \alpha_{k_1},\ u \oplus w,\ \beta_2, \ldots, \beta_{k_2}\big)$$

leads from $(y\|x)$ to $(w'\|x)$. Hence $L(y)$ is a $(k_1 + k_2)$-block bridge at capacity x from rate y, and $L(w')$ is closed. For $k_1 = k_2 = 1$ this is Theorem 1.

(ii) Cycles of the capacity map. *Let $h(z) = \operatorname{cap} f(0^r\|z)$ and let $z_0, z_1, \ldots, z_{\ell-1}$ be a cycle of h of length $\ell \geq 1$ (indices modulo ℓ), with $f(0^r\|z_t) = (q_t\|z_{t+1})$. Then for every y the ℓ-block word $(y, q_0, \ldots, q_{\ell-2})$ leads from $(y\|z_0)$ to $(q_{\ell-1}\|z_0)$, and it is closed for $y = q_{\ell-1}$. Likewise, if $z_0, \ldots, z_{\ell-1}$ is a cycle of $h'(z) = \operatorname{cap} f^{-1}(0^r\|z)$ with $f^{-1}(0^r\|z_t) = (p_t\|z_{t+1})$, then $(y \oplus p_{\ell-1}, p_{\ell-2}, \ldots, p_0)$ leads from $(y\|z_0)$ to $(0^r\|z_0)$, and it is closed for $y = 0^r$.*

(iii) Second preimages. *Let B be any bridge from (i) or (ii) at capacity x from rate y_0, with end state $(y'\|x)$ and closed companion B' (Lemma 1), let $0^b \xrightarrow{C_0} (y_0\|x)$ and $(y'\|x) \xrightarrow{C_1} (y_1\|0^c)$. Then for every message M of at least r bits, with padded blocks $M_0 \cdots M_n$, the padded sequences*

$$C_0 \,\|\, B \,\|\, (B')^i \,\|\, C_1 \,\|\, (M_0 \oplus y_1) \,\|\, M_1 \cdots M_n, \qquad i \geq 0,$$

are valid paddings of pairwise distinct messages with the same digest as M for every output length.

Proof. (i) From $(y\|x)$, absorbing $y \oplus a$ gives $f(a\|x) = T_1$, and absorbing $\alpha_2, \ldots, \alpha_{k_1}$ gives $T_{k_1} = (u\|v)$ by the definition of the T_t. Absorbing $u \oplus w$ gives $f(w\|v) = f(w\|v') = f(f^{-1}(Z_1)) = Z_1$. For $2 \leq t \leq k_2$, if the state is Z_{t-1}, absorbing β_t gives $f\big(Z_{t-1} \oplus (\beta_t\|0^c)\big) = f(f^{-1}(Z_t)) = Z_t$; by induction the state after the last block is $Z_{k_2} = (w'\|x)$. The start and end capacities are both x, so $L(y)$ is a bridge in the sense of Definition 2, and with $y = w'$ the rate offset is 0^r. (ii) From $(y\|z_0)$, absorbing y gives $f(0^r\|z_0) = (q_0\|z_1)$. If the state is $(q_{t-1}\|z_t)$, absorbing q_{t-1} gives

$f(0^r\|z_t) = (q_t\|z_{t+1})$. After ℓ blocks the state is $(q_{\ell-1}\|z_\ell) = (q_{\ell-1}\|z_0)$. For h': from $(y\|z_{t+1})$ the block $y \oplus p_t$ gives $f(p_t\|z_{t+1}) = f(f^{-1}(0^r\|z_t)) = (0^r\|z_t)$; starting at $(y\|z_0) = (y\|z_\ell)$ and applying this for $t = \ell-1, \ell-2, \ldots, 0$ (with rate y for the first block and 0^r afterwards) ends in $(0^r\|z_0)$. (iii) By Lemma 1, $B\|(B')^i$ leads from $(y_0\|x)$ to $(y'\|x)$ for every $i \geq 0$. The rest is the proof of Proposition 3: after C_1 the state is $(y_1\|0^c)$, the block $M_0 \oplus y_1$ gives $f(M_0\|0^c)$, and the blocks $M_1, \ldots, M_n$ are those of M. Since M_0 consists of message bits only, each sequence is the padding of the message obtained by replacing the first r bits of M; its length differs from $|M|$ by a multiple of r, so the padding is unchanged. The sequences have pairwise different lengths, hence are distinct. □

Remark 5 (Cost of Theorem 2)**.** The statements are exact; the costs are heuristic for an ideal permutation. In (i) one fixes all blocks but α_{k_1} and β_2 and varies only these two, so each forward candidate costs one call of f and each backward candidate one call of f^{-1} (if $k_1 = 1$ or $k_2 = 1$, one varies a or w' instead); the capacity match $v' = v$ then costs $\approx 2 \cdot \frac{\sqrt{\pi}}{2} 2^{c/2} \approx 1.77 \cdot 2^{c/2}$ calls plus $k_1 + k_2 - 2$ calls for the fixed parts, the same order as Theorem 1 (a closed loop with $k_1 = k_2 = 3$ and second preimages by (iii) are checked by `03_campaigns/bridge_cycle_search/kblock_loop.py`); longer loops give more freedom in the blocks but do not go below $2^{c/2}$. In (ii) no matching is needed: a memoryless cycle finder (Brent) applied to h or h' from any start finds a cycle after $\Theta(2^{c/2})$ evaluations for a random mapping (the first repeated state occurs after $\approx \sqrt{\pi 2^c/2} \approx 1.25 \cdot 2^{c/2}$ steps; the evaluation count of a particular cycle finder has a different constant [20, §4]), but a typical cycle has length $\ell \approx \sqrt{\pi 2^c/8} \approx 0.63 \cdot 2^{c/2}$, so the bridge itself has $\Theta(2^{c/2})$ blocks; short cycles such as the 286-cycle of Appendix E are rare. In (iii) the connectors C_0, C_1 cost $\approx \sqrt{\pi} \cdot 2^{c/2}$ each, as in Proposition 3; the 2011 bridge ($x = 0^c$) needs no connectors and remains the cheapest route to second preimages. After this precomputation no further search is needed; writing out and hashing the i-th message costs $|C_0| + (i+1)|B| + |C_1| + n + 1$ calls of f, i.e. time linear in its length. Part (i) is the multi-block form of the path-to-inner-state search of [12, §5.5], and part (ii) is the classical rho structure of a random mapping applied to the capacity map; we state them as a theorem to make the bridges and their costs explicit, not as new attacks.

Remark 6 (Examples at full rounds)**.** Proposition 3 and Theorem 1 have been instantiated for the *full* 18-round Keccak-f[200] and the full 24-round Keccak-f[1600], each with $c = 32$ (hex, Keccak byte order). For f[200]:

Proposition 3 (single-state bridge)	
S	`ebf75bbca4aad534a35f8bb6d77a7ec0d37c265e91d18cd35a`
P_1	`48c6344fa6f6c8a943a42f426fbc76e4b7714bd21d`
P_3	`de3283adf6fabd00a2508529f404220ba14738f6d0`
x	`2e62e846`
Theorem 1 (birthday loop)	
x	`81a8ab9b`
a	`4f8c786c44065e01a8acb3b3e820530b1fb8b25edd`
w'	`caa72c0d46859b3c7e2ce32cafe9e1104835ff59bf`
$u \oplus w$	`53fbcd88de6ff9683217c6d2240db17133350f4902`

The measured costs on 8 cores match the analysis. The single-state bridge took $2.34 \cdot 10^9$ calls each of f and f^{-1} ($0.55 \cdot 2^c$, 413 s) for f[200] and $5.74 \cdot 10^9$ ($1.34 \cdot 2^c$, 1716 s) for f[1600]. The birthday loop took $1.5 \cdot 10^4$ (f[200]) and $8.7 \cdot 10^4$ (f[1600]) calls per side, below one second, and each connector C_0, C_1 took 2^{16} forward and at most $1.3 \cdot 10^5$ backward calls; the connector C_0 from the all-zero IV to $(y_0\|x)$ is the measured IV transition of each of these examples. For every example the messages with $i = 0, 1, 2$ were checked against a two-block SHA-3-padded M: same final state and same digest, using XKCP for f[1600] and an independent bit-level f[200] reference. The full vectors (C_0, C_1, y_0, y_1, M, and the f[1600] instances) are in Appendix D and

the artifact (`02_bridges_fig1/general_bridge_examples/`). A step-by-step worked example with a concrete message, a 64-bit output and a short second preimage is in Section 2.4.

Remark 7 (A full-round bridge with $c = 80$)**.** On the full 24-round Keccak-$f[1600]$ with $r = 1520$ and $c = 80$ we found the following three rate blocks of 190 bytes each (hex, FIPS 202 byte order: rate bytes 0..189, capacity bytes 190..199). Starting from the all-zero state, XOR P_1, apply f, XOR M, apply f, XOR P_3 gives the all-zero 1600-bit state, so (P_1, M, P_3) takes the sponge from capacity 0^{80} back to 0^{80} (the intermediate capacity is `75f1c14065ec94efa875`). Since $x = 0^{80}$ is the capacity of the standard IV, this example needs no IV transition; for a bridge at a nonzero capacity with a measured standard-IV transition, see Section 3.4.1.

```
P1 = 1cc82c1912d6eb74342831943dd9ea069daec34ee902e58143f5a1c7cdb7b7aa
     6d30f97ecf5477f23edb3c58d83daeb85d8ed99dda3e939941d33dcbb4249a32
     44212edf98cfcf358398db739e75f241594939a550419b8e09e2bb09dee94100
     58f481d3586f00c21e3f359c82f13c0a3e42cb91531145a7083303d5fef857d2
     44ecbd1e457aefafd02961e9cfb7700ff2c2287fc96ed14eec3a1107c66b2ba1
     592d8584f5439afa0e97772e7371e381fdac9f68a339f35b74986249f06e
M  = 47650232e4942ee92ce1fb61ea15912f303d106c79c14de79dde15fbdf8465fb
     23c22005c0ee6c5b7c6bb6ddddbe6c4eceae7d8bd46ac9179e0b8131dda3f3a4
     ce247f43e1d2a85803c6301888d9ca30531f1dbaddf6cc9a2ff11e239f8ad646
     9c679393efe4613b7f7422ecedc413e5775468723e8eb76b3ccafa8573530cfd
     e5f9a2ddad2d57b2713889848d8c381e82b32670eaea2218ba0999140883e787
     33937b3772c9a385451bf7ac2009c154ed02763b6497af606b3bf0b4cd0c
P3 = 4e83dfbaff6ee5fe7db7227636e9ad5a235bedde2cf2e2f60284fd35f83c8952
     eab9a134f7bd74655f854eaf22417367cb762671e1d0bb7031fdbeac9d8ee279
     25096c0747f6b1f0c04756eebe751413479f8827ac77fb6073f9dc5c03053b14
     fedbf376de1f85bde3b504f7bb4b2aa651a88142b4fbe6d983212eef2ec4f5cc
     d6aecc93ebff845db88a725d4118780144ccaf26b84e1888f7357a25f4da5e6c
     2dcc09b86fb5d3e8b92ffa65527048c52a4cf34ef241beb85e3eee0396a7
```

We found it on 30.10.2026 with the parallel distinguished-point (van Oorschot–Wiener [73]) search with $d = 20$ and a complete distinguished-point table ($2^{21.41}$ stored points, no lost claws), after $2^{41.41}$ Keccak-f calls, in about 7.5 h wall-clock on three machines (a laptop and two servers, about $2^{26.70}$ calls/s combined). The heuristic generic mean for this search is $\sqrt{\pi} \cdot 2^{c/2} = 2^{40.83}$ calls, so the run took about 1.5 times the mean (probability about 0.17 of taking at least that long; see Remark 11 for detection latency and a combined test over four $c = 80$ runs). An earlier version of this remark compared the run with a budget of $3 \cdot 2^{40}$, based on a measured constant of $2.8 \cdot 2^{c/2}$; that constant was an artifact of a bounded distinguished-point table that overwrites entries, and with a complete table the constant matches $\sqrt{\pi}$: over $2^{10.54}$ (1485) verified bridges at $c = 40$, 44 and 48 with the same walk and identical parameters, the measured cost is 0.965 ± 0.047 of the model for Keccak-$f[1600]$, 0.976 ± 0.050 for a lazily sampled true random permutation and 0.979 ± 0.049 for a 4-round SHAKE256 Feistel network (95% intervals; artifact folder `08_rp_control/`). We therefore see no distinguisher of Keccak in this constant. The bridge was verified independently by a standalone Python Keccak and by an XKCP-based verifier (artifact folder `bridges/c80/`), and it is the first solved instance of the public challenge of Section 2.7.

2.4 A worked example

We follow Theorem 1 step by step on the full 18-round Keccak-$f[200]$ with $c = 32$, $r = 168$ (21 bytes), using the birthday-loop instance of Remark 6. All values are hex in Keccak byte order; a state is written as rate | capacity, and T_j is the state after the j-th absorbed block. Every number below is printed and checked by `worked_example/worked_example.py` in `02_bridges_fig1/general_bridge_examples/`, which evaluates each call of f with the bit-level FIPS 202 reference (checked against the XKCP Keccak-$f[200]$ test vectors) and with a second, lane-based implementation (see Zenodo [33]).

Ingredients. The capacity match (1) and the two connectors are given by

x	`81a8ab9b`	capacity of the loop
a	`4f8c786c44065e01a8acb3b3e820530b1fb8b25edd`	$f(a\|\|x) = (u\|\|v)$
u, v	`62ec63fb5851a6e6a62fa0bdff816a9de51b66d73c`, `abcf4a9d`	
w'	`caa72c0d46859b3c7e2ce32cafe9e1104835ff59bf`	$f^{-1}(w'\|\|x) = (w\|\|v)$
w	`3117ae73863e5f8e9438666fdb8cdbecd62e699e3e`	
y_0	`c93e48e6aa432b28061046bba22273cce4572d86bc`	$0^b \xrightarrow{C_0} (y_0\|\|x)$
y_1	`2f93ecdbce4b4c5c89ea775df40aa984046dee0e2b`	$(w'\|\|x) \xrightarrow{C_1} (y_1\|\|0^c)$

where the relation for f^{-1} is checked as $f(w\|\|v) = (w'\|\|x)$. The connector blocks $C_0 = (a_1, a_2)$ and $C_1 = (d_1, d_2)$ are in Table 1. Neither connector depends on the message, so they were not recomputed.

The message M. We take the 24-byte ASCII string $M =$ "`Beyond the Event Horizon`". With pad10*1 and the SHA-3 domain suffix (first padding byte `06`, last byte `80`), it gives the two blocks M_0M_1 ($k = 1$, as Theorem 1 requires) shown in the upper part of Table 1. The output is the first 64 bits of the rate of the final state (64 bits $< r$, so no further call of f is needed):

$$H_{64}(M) = \texttt{130c66d7cbf4e20f}.$$

The second preimages. The lower part of Table 1 absorbs the 10-block second preimage ($i = 1$)

$$M'_1 = C_0 \,\|\, L(y_0) \,\|\, L(w') \,\|\, C_1 \,\|\, (M_0 \oplus y_1) \,\|\, M_1.$$

Steps 1–2 reach $(y_0\|x)$. Step 3 absorbs $y_0 \oplus a$, so the input of f is $(a\|x)$ and the result is $(u\|v)$. Step 4 absorbs $u \oplus w$, so the input is $(w\|v) = f^{-1}(w'\|x)$ and the result is $(w'\|x)$. Steps 5–6 are the loop $L(w')$: the inputs of f are again $(a\|x)$ and $(w\|v)$, and the state returns to $(w'\|x)$. Steps 7–8 reach $(y_1\|0^c)$, the capacity with all zero bits. In step 9 the block $M_0 \oplus y_1$ cancels y_1, so the input of f is $(M_0\|0^c)$, exactly as in step 1 of M; from here on both messages absorb the same states, and $H_{64}(M'_1) = H_{64}(M)$ (the full 200-bit states agree as well). The shortest member of the family is M'_0 ($i = 0$): blocks 1–4 and 7–10 of the table, 8 padded blocks, i.e. 150 message bytes (the 147 bytes of the first 7 blocks followed by "`zon`"), again with $H_{64}(M'_0) = H_{64}(M)$. Each further copy of $L(w')$ (steps 5–6) adds two blocks; the script also checks $i = 2$. Figure 2 shows the path.

2.5 Consequences

Second and extra preimages. One bridge B at the capacity with all zero bits from rate 0^r, with rate offset δ and closed companion B' (Lemma 1), gives, for every message $M = M_0 \cdots M_k$ of at least r bits, the messages $B\|(B')^i\|(\delta \oplus M_0)\|M_1 \cdots M_k$, $i \geq 0$, with the same digest, with no further search (writing and hashing each message costs time linear in its length) (for the 2011 bridge, $B = (P_1, P_2 \oplus P_4)$ and $\delta = P_3$, which is the family of Section 2); the whole cost is finding one bridge. Combined with a known first preimage (e.g. the 26-block 3-round Keccak-256 preimage of Lin–He–Yu at $\approx 2^{57.2}$ [57]) this gives unboundedly many further preimages; the first preimage does not lower the bridge cost.

Contrast with classical collisions. A classical collision search, for example the birthday search on the n-bit digest, outputs one pair (M, M') *chosen by the search*. It yields no second preimage for a message chosen by someone else. In the hash-based setting where only one preimage of a collision is published, the secret second message is likewise tied to that single pair. A second preimage for a *given* message still costs $\approx 2^n$ generically for an ideal n-bit hash. The Fig. 1 bridge is different. One precomputation of $\approx 2^{c/2}$ calls, independent of any target,

Table 1: Worked example of Theorem 1 on the full Keccak-$f[200]$, $c = 32$. Every step is one call of f (no f^{-1} is evaluated when hashing). For each step j the first line gives the name of the absorbed block and the input of f, the second line the block in hex and the result T_j. Top: the message $M =$ "`Beyond the Event Horizon`". Bottom: the second preimage M'_1; M'_0 omits steps 5–6.

j	block m_j (name, then hex)	top: input $T_{j-1} \oplus (m_j \| 0^c)$ of f; bottom: result T_j	(rate | capacity)
Message $M = M_0 M_1$, from $T_0 = 0^b$			
1	M_0	`4265796f6e6420746865204576656e7420486f7269`	`00000000`
	`4265796f6e6420746865204576656e7420486f7269`	`5a50918ab5217d1490eee80d1d5d449adbbf01b516`	`7b65c9ae`
2	M_1	`203fff8cb5217d1490eee80d1d5d449adbbf01b596`	`7b65c9ae`
	`7a6f6e0600000000000000000000000000000080`	`130c66d7cbf4e20f03149ab9ac4c6c4ef7a55d9a04`	`872231fa`
Second preimage $M'_1 = C_0 \| L(y_0) \| L(w') \| C_1 \| (M_0 \oplus y_1) \| M_1$, from $T_0 = 0^b$			
1	a_1 (C_0)	`b6267748af70a1eff3123c7a38ceac3c43b2318a6c`	`00000000`
	`b6267748af70a1eff3123c7a38ceac3c43b2318a6c`	`6edde52ee642008ff35d26f5ba7e6cdc1e496102ca`	`f1534d23`
2	a_2 (C_0)	`988e88d3e090d4693f6d543a62d12cce84307f8f15`	`f1534d23`
	`f6536dfd06d2d4e6cc3072cfd8af40129a791e8ddf`	`c93e48e6aa432b28061046bba22273cce4572d86bc`	`81a8ab9b`
3	$y_0 \oplus a$	`4f8c786c44065e01a8acb3b3e820530b1fb8b25edd`	`81a8ab9b`
	`86b2308aee457529aebcf5084a0220c7fbef9fd861`	`62ec63fb5851a6e6a62fa0bdff816a9de51b66d73c`	`abcf4a9d`
4	$u \oplus w$	`3117ae73863e5f8e9438666fdb8cdbecd62e699e3e`	`abcf4a9d`
	`53fbcd88de6ff9683217c6d2240db17133350f4902`	`caa72c0d46859b3c7e2ce32cafe9e1104835ff59bf`	`81a8ab9b`
5	$w' \oplus a$	`4f8c786c44065e01a8acb3b3e820530b1fb8b25edd`	`81a8ab9b`
	`852b54610283c53dd680509f47c9b21b578d4d0762`	`62ec63fb5851a6e6a62fa0bdff816a9de51b66d73c`	`abcf4a9d`
6	$u \oplus w$	`3117ae73863e5f8e9438666fdb8cdbecd62e699e3e`	`abcf4a9d`
	`53fbcd88de6ff9683217c6d2240db17133350f4902`	`caa72c0d46859b3c7e2ce32cafe9e1104835ff59bf`	`81a8ab9b`
7	d_1 (C_1)	`814d77d778dd4fd48063981d7f48e44a49def0a918`	`81a8ab9b`
	`4bea5bda3e58d4e8fe4f7b31d0a1055a01eb0ff0a7`	`5e038a39e4306cc270236432b2c0ed265627eb5213`	`5ca24cb5`
8	d_2 (C_1)	`4eb6e63afc806ab489843077a509a3be8920fbb619`	`5ca24cb5`
	`10b56c0318b00676f9a7544517c94e98df0710e40a`	`2f93ecdbce4b4c5c89ea775df40aa984046dee0e2b`	`00000000`
9	$M_0 \oplus y_1$	`4265796f6e6420746865204576656e7420486f7269`	`00000000`
	`6df695b4a02f6c28e18f5718826fc7f02425817c42`	`5a50918ab5217d1490eee80d1d5d449adbbf01b516`	`7b65c9ae`
10	M_1	`203fff8cb5217d1490eee80d1d5d449adbbf01b596`	`7b65c9ae`
	`7a6f6e0600000000000000000000000000000080`	`130c66d7cbf4e20f03149ab9ac4c6c4ef7a55d9a04`	`872231fa`
Squeeze: $H_{64}(M) = H_{64}(M'_1) = H_{64}(M'_0) =$ first 64 bits of the rate of $T_{\text{last}} =$ `130c66d7cbf4e20f`			

yields second preimages of *every* message M of at least r bits, including messages chosen after the bridge was found, with no further search (writing and hashing each message costs time linear in its length). The precomputation is universal and is amortized over all targets. This is consistent with the sponge's known generic second-preimage bound of $\min(2^n, 2^{c/2})$ [12]. It is the constructive, reusable form of that bound, and it does not break Keccak: at the standard SHA3-256 capacity $c = 512$, the precomputation costs 2^{256}.

Cheap collisions and multicollisions.

Theorem 3 (Cheap collisions and bridge-amplified multicollisions)**.** *Let f be a permutation of $\{0,1\}^b$, $b = r + c$.*

(i) Cheap collisions. *Let B be a bridge at the capacity with all zero bits from rate 0^r, with $0^b \xrightarrow{B} (\delta \| 0^c)$ (for the 2011 bridge, $B = (P_1, P_2 \oplus P_4)$ and $\delta = P_3$). For every message M of at least r bits, with padded blocks $M_0 \cdots M_n$, the sequence $B \,\|\, (\delta \oplus M_0) \,\|\, M_1 \cdots M_n$ is the padding of a message $\widehat{M} \neq M$ with the same digest as M for every output length. Moreover, $(B \,\|\, \delta)$ and the single block 0^r lead from 0^b to the same full state $f(0^b)$, so every such bridge yields an inner collision with no further call of f.*

(ii) Multicollisions. *Let $A \neq A'$ be block sequences of the same length $\lambda \geq 1$ with $0^b \xrightarrow{A} s_1$ and $0^b \xrightarrow{A'} s_1$ for a full state s_1, and let B be a block sequence of length μ with $s_1 \xrightarrow{B} (\delta \| 0^c)$ for some $\delta \in \{0,1\}^r$ ($\delta = 0^r$ is allowed). Let A_δ, A'_δ be A, A' with δ XOR-ed into their*

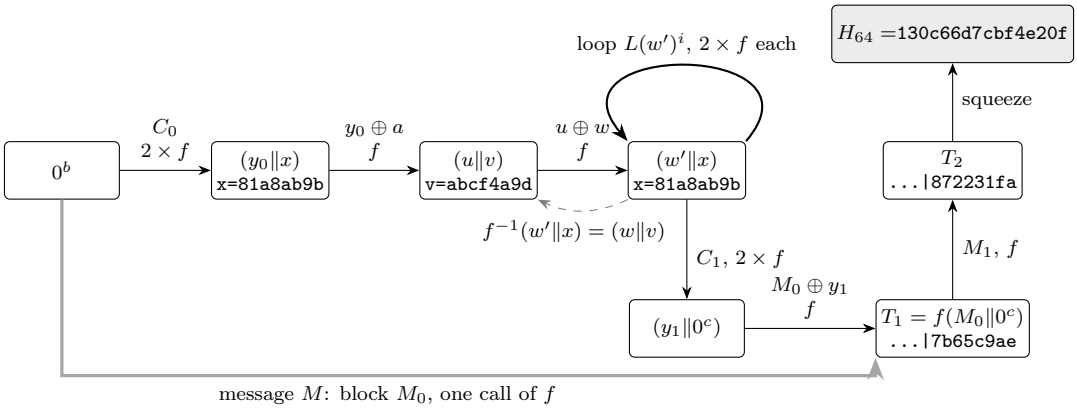

Figure 2: Path of the worked example (Table 1). Upper route with the loop: the second preimages $M'_i = C_0 \| L(y_0) \| L(w')^i \| C_1 \| (M_0 \oplus y_1) \| M_1$; lower grey route: the message $M = M_0 M_1$. Both reach $T_1 = f(M_0 \| 0^c)$, hence the same T_2 and the same 64-bit output. The dashed arrow is the backward relation used to find w; hashing evaluates only f.

first block. For $k \geq 1$ and $\mathbf{e} = (e_1, \ldots, e_k) \in \{0,1\}^k$ put $X_1 = A$ if $e_1 = 0$ and $X_1 = A'$ otherwise, $X_j = A_\delta$ if $e_j = 0$ and $X_j = A'_\delta$ otherwise ($j \geq 2$), and

$$W_{\mathbf{e}} = X_1 \,\|\, B \,\|\, X_2 \,\|\, B \,\|\, \cdots \,\|\, B \,\|\, X_k.$$

Then the 2^k words $W_{\mathbf{e}}$ are pairwise distinct, all have $k\lambda + (k-1)\mu$ blocks, and $0^b \xrightarrow{W_{\mathbf{e}}} s_1$ for every $\mathbf{e}$. Consequently, for every message S (of any length, possibly empty), the 2^k messages $W_{\mathbf{e}} \| S$ are distinct, have equal length and have the same digest for every output length.

Proof. (i) From 0^b, B leads to $(\delta \| 0^c)$, and absorbing $\delta \oplus M_0$ gives $f(M_0 \| 0^c)$, the state after M_0 from 0^b; the blocks $M_1, \ldots, M_n$ coincide. Since $|M| \geq r$, the block M_0 consists of message bits only, so the sequence is the padding of the message $\widehat{M}$ obtained by replacing the first r bits of M by $B \| (\delta \oplus M_0)$, as in the proof of Proposition 3; $\widehat{M} \neq M$ because it is longer. Absorbing δ after B gives $f(0^b)$, which is also the state after the single block 0^r, and $B \| \delta$ has at least two blocks, so it differs from 0^r. (ii) If the first block of A is m_1, then absorbing $m_1 \oplus \delta$ into $(\delta \| 0^c)$ gives $f(m_1 \| 0^c)$, the state after m_1 from 0^b; the remaining blocks of A_δ and A coincide, so $(\delta \| 0^c) \xrightarrow{A_\delta} s_1$, and likewise for A'_δ. By induction on j: after X_1 the state is s_1; if it is s_1 after X_j, then B leads to $(\delta \| 0^c)$ and X_{j+1} back to s_1. So every $W_{\mathbf{e}}$ leads from 0^b to s_1, and the block count follows from $|X_j| = \lambda$. For distinctness let $\mathbf{e} \neq \mathbf{e}'$ and let j be the first index with $e_j \neq e'_j$. Because all X have length λ and B is fixed, the j-th slots of $W_{\mathbf{e}}$ and $W_{\mathbf{e}'}$ occupy the same block positions, and they differ: $A \neq A'$, and XOR-ing the same δ into the first block is injective, so $A_\delta \neq A'_\delta$. Finally, each $W_{\mathbf{e}}$ has a length that is a multiple of r, so the padding of $W_{\mathbf{e}} \| S$ is $W_{\mathbf{e}}$ followed by the padded blocks of S, which are the same for all $\mathbf{e}$; all states agree after $W_{\mathbf{e}}$, hence after the whole padded message. □

Remark 8 (Cost of Theorem 3; relation to Joux)**.** The statements are exact; the costs are heuristic for an ideal permutation. In (i) the only search is the bridge ($\approx 2 \cdot 2^{c/2}$ calls for the 2011 bridge); since a bridge yields an inner collision for free, finding it is at least as hard as an inner collision. In (ii) the setup is one inner collision and one connector B, independent of k: a capacity collision among one-block messages, $\operatorname{cap} f(m\|0^c) = \operatorname{cap} f(m'\|0^c)$, costs $\approx \sqrt{\pi/2} \cdot 2^{c/2} \approx 1.25 \cdot 2^{c/2}$ calls of f and gives $A = (m, 0^r)$, $A' = (m', \operatorname{rate} f(m\|0^c) \oplus \operatorname{rate} f(m'\|0^c))$ with $\lambda = 2$; the two-block connector from $\operatorname{cap} s_1$ to the capacity with all zero bits costs $\approx 1.77 \cdot 2^{c/2}$ calls of f and f^{-1}, with $\mu = 2$. The total is $\approx 3 \cdot 2^{c/2}$, against $\approx 1.25\,k \cdot 2^{c/2}$ for the k collision

searches of Joux's construction [46]. No further search is needed for any k, but writing out and hashing one message costs $k\lambda + (k-1)\mu$ calls of f plus those for S, and listing all 2^k messages costs 2^k times as much. The construction is Joux's, with one sponge-specific observation: a connector back to the capacity with all zero bits returns the sponge to its starting state up to the rate, so a single collision can be reused in every slot. Multicollisions whose cost does not depend on their size were already given for CubeHash by Aumasson et al. [3, §3] (2^{64}-collisions within 2^{65} calls, against 2^{70} for Joux's method), and are also described by Ferguson, Lucks and McKay [29, §1.3]; they rely on the full-state zero fixed point of CubeHash, and their colliding messages differ only in length; for generic sponges, [12, §5.10.1] states the cost $s \cdot 2^{(c+3)/2}$ of a 2^s-fold multicollision, linear in s, and the reusable path to 0^c appears in [12, §5.10.2] and [34] only for second preimages. Theorem 3(ii) is the equal-length version for generic sponges with a c-bit inner state. To our knowledge, Theorem 3(ii) is the first construction of equal-length 2^k-multicollisions for generic sponges whose search cost ($\approx 3 \cdot 2^{c/2}$) is independent of k: a single inner collision and a single connector back to the same capacity are reused in every slot. The gain is in search cost only, a factor of about $k/2.4$ over Joux's construction, which grows with k but does not change the exponent; it does not go below the generic $2^{c/2}$ inner-collision bound.

Both code paths satisfy the hypotheses of Theorem 3(ii): `gen.py` (Section E.4) uses $\lambda = \mu = 2$ and $\delta = P_3$, and `mc.c` on $f[200]$ uses $\lambda = 2$ and a long connector that returns to 0^b exactly ($\delta = 0^r$), found by meeting a forward walk of h from s_1 with a backward walk of h' from $f^{-1}(0^b)$. Verified on $f[200]$ for $(c, R) \in \{24, 32\} \times \{2, 4\}$ with $k = 10$: all 1024 messages absorb to the identical state; Joux/bridge-amplified call ratios 4.13 ($c24, R4$), 5.78 ($c24, R2$), 6.10 ($c32, R4$), 6.02 ($c32, R2$), i.e. savings of the order of $k/2$, consistent with Remark 8 up to the constants of the two connector searches. No sub-generic claim. *Contrast with classical multicollisions.* For an ideal n-bit hash, a t-fold multicollision costs $\approx 2^{n(t-1)/t}$. Joux's construction on iterated hashes needs k separate collision searches for 2^k messages, i.e. $\approx k \cdot 2^{c/2}$ for a sponge. In both cases the cost grows with the size of the multicollision. The bridge-amplified construction pays $\approx 3 \cdot 2^{c/2}$ once (one collision and one connector, Remark 8) and then gives 2^k equal-length colliding messages for *any* k at no further search cost (the output size grows with k). A bridge B at the capacity with all zero bits from rate 0^r additionally gives, for every message M of at least r bits, the infinite family $B \| (B')^i \| (\delta \oplus M_0) \| M_1 \cdots M_k$ ($i \geq 0$) of colliding messages of different lengths. As with second preimages, the precomputation is universal and amortized, and it is independent of any target. The costs are generic.

2.6 Search algorithms

Write $x_k = \operatorname{cap} f(S_k)$, $y_k = \operatorname{cap} f^{-1}(S_k)$ and $P_{1,k} = \operatorname{rate} f(S_k)$, $P_{3,k} = \operatorname{rate} f^{-1}(S_k)$.

Algorithm 1 Literal bridge loop (reference)

$S_0 \leftarrow 0^r \| 0^c$ (or a seeded start)
for $k = 0, 1, 2, \ldots$ **do**
  $(P_1 \| x) \leftarrow f(S_k)$; $(P_3 \| y) \leftarrow f^{-1}(S_k)$
  if $y = x$ **then return** single-state bridge S_k, $\Delta = P_1 \oplus P_3$
  end if
  $S_{k+1} \leftarrow P_3 \| x$ ▷ rate from the inverse side, capacity from the forward side
end for

Each step succeeds with probability 2^{-c} for random-like f, so the waiting time is geometric, mean 2^c, median $\ln 2 \cdot 2^c$. A cycle detector (Brent, $O(1)$ memory) is included; the rate-fixed variant $S_{k+1} = \operatorname{rate}(S_0) \| x_k$ is the one where cycles appear. After n steps there are n^2 cross pairs, so $\Pr[\text{no hit}] \approx e^{-n^2/2^c}$, giving $\mathbb{E}[k] = \frac{\sqrt{\pi}}{2} 2^{c/2} \approx 0.886 \cdot 2^{c/2}$ and median $\sqrt{\ln 2}\, 2^{c/2} \approx 0.833 \cdot 2^{c/2}$. (random-function model). Memory is $\Theta(2^{c/2})$ entries (8 bytes each in our C code).

Algorithm 2 Birthday connector search (two tables)

$S_0 \leftarrow$ start; $X \leftarrow \emptyset$, $Y \leftarrow \emptyset$
for $k = 0, 1, 2, \ldots$ **do**
  $(P_{1,k} \| x_k) \leftarrow f(S_k)$; $(P_{3,k} \| y_k) \leftarrow f^{-1}(S_k)$
  insert $x_k \mapsto k$ into X, $y_k \mapsto k$ into Y
  if $\exists\, i, j \le k :\ x_i = y_j$ **then**
    replay $S_0..S_k$; **return** connector $(S_i, S_j, \Delta = P_{1,i} \oplus P_{3,j})$ ▷ Prop. 2; a bridge iff $\operatorname{cap} S_i = \operatorname{cap} S_j$
  end if
  $S_{k+1} \leftarrow P_{3,k} \| x_k$
end for

Connectors versus bridges. A hit of Algorithm 2 is a two-point connector: the word $(0^r, \Delta)$ leads from S_i to S_j, but in general $\operatorname{cap} S_i \neq \operatorname{cap} S_j$, so it is *not* a bridge in the sense of Definition 2. For random-like f the extra condition $\operatorname{cap} S_i = \operatorname{cap} S_j$ holds with probability 2^{-c} per hit, so requiring it in Algorithm 2 would bring the cost back to $\approx 2^c$. For example, the full-round $c = 32$ hit at $i = 10846$, $j = 31456$ has $\operatorname{cap} S_i =$ `fcccb460` $\neq \operatorname{cap} S_j =$ `191fc96b`, although $\operatorname{cap} f(S_i) = \operatorname{cap} f^{-1}(S_j)$. Our code now prints this endpoint check separately from the intermediate match (line `TYPE endpoint_capacity_equal`). A bridge at birthday cost is obtained instead by fixing the capacity x on both sides, as in Theorem 1: random a and w' are drawn, $\operatorname{cap} f(a\|x)$ and $\operatorname{cap} f^{-1}(w'\|x)$ are stored in two tables, and the first match gives the closed bridge $L(w')$ at capacity x. Both endpoints then have capacity x by construction, and the expected cost is $2 \cdot \frac{\sqrt{\pi}}{2} 2^{c/2} \approx 1.77 \cdot 2^{c/2}$ calls of f_R and f_R^{-1} together (median $\approx 1.67 \cdot 2^{c/2}$). We measured this search on $f[200]$ for $R = 1..18$, 5 seeds each (`mitm_fixed_cap.py`, 90 runs per c): the mean (median) cost was 1.91 (1.83), 1.83 (1.78) and 1.49 (1.40)$\cdot 2^{c/2}$ for $c = 16, 20, 24$. A second set of 90 runs at $c = 24$ gave 1.73 (1.50), consistent with sampling variation (the per-run standard deviation is about half the mean). Every hit was checked to give $(w'\|x) \xrightarrow{L(w')} (w'\|x)$. The distinguished-point search below matches capacities that start from the capacity with all zero bits on both sides, so its outputs are bridges in the sense of Definition 2 (the 2011 form).

Distinguished points. For $c \ge 64$ the tables exceed the available memory (c=64 needs $\ge 2^{32}$ steps and 32+ GB). We therefore implemented the memoryless alternative, a van Oorschot–Wiener parallel collision search [73] on capacity values (`02_bridges_fig1/tmto_dp/pcs.c`). A selector bit $s(x)$ defines $g(x) = \operatorname{cap} f_R(P_1(x)\|0^c)$ if $s(x) = 0$ and $g(x) = \operatorname{cap} f_R^{-1}(P_3(x)\|0^c)$ otherwise; trails $x \mapsto g(x)$ run until a distinguished point (the low d bits are zero, $d = c/2 - \log_2 M$ for a table of M slots) and only (endpoint, start, length) is stored. Collisions with differing selector bits are claws (bridges); about half are same-side and are kept as free inner collisions; false alarms are removed by lockstep recomputation and full absorption. On $f[200]$, $R \in \{2, 4, 18\}$, $c \in \{24, 32, 40\}$, the DP search cost stayed ≈ 2–$4 \cdot 2^{c/2}$ across memory $2^8..2^{16}$, with useful-claw fraction 0.50 ± 0.08. A proof of concept with full $R = 18$ rounds (single-threaded runs, four in parallel) found a bridge in every run: 20/20 for $c = 24$ and $c = 32$, 8/8 for $c = 40$ and 4/4 for $c = 48$ ($d = 4, 6, 8, 10$; $M = 2^8, 2^{10}, 2^{12}, 2^{14}$). The mean cost was 3.8, 3.0, 1.9 and $2.8 \cdot 2^{c/2}$ calls of f_R or f_R^{-1} (individual runs 0.2–9.9 $\cdot 2^{c/2}$). For $c = 48$ the DP table needs 512 KiB instead of about 256 MiB for a plain table of 2^{24} entries, and one run takes about 2.5 minutes on one core. All 52 bridges pass two independent checkers (C and Python: return to 0^{200} and second preimages under three paddings); they are bridges in the sense of Definition 2 at the capacity with all zero bits (the 2011 form), so verified bridges reach $c = 48$ on the full 18-round $f[200]$. The cost constants are empirical, from small samples, and assume that g behaves like a random function. These runs use a bounded table of M slots that overwrites entries, which loses claws and inflates the constant; with a complete table the mean

cost is $\approx \sqrt{\pi} \cdot 2^{c/2} \approx 1.77 \cdot 2^{c/2}$ (Remark 7).

Parallel version. A multithreaded variant (see Zenodo [33] `02_bridges_fig1/tmto_dp/mt/pcs_mt.c`) lets T threads share one distinguished-point table, so the total number of calls stays ≈ 2–$4 \cdot 2^{c/2}$ while the wall time drops by about T. It supports both $f[200]$ and the full 24-round Keccak-$f[1600]$ with $c \leq 128$, and a deterministic mode reproduces every run from its seed. On $f[200]$, $R = 18$ (16 runs per cell, 8-core machine, one job at a time), the throughput grew from $3.3 \cdot 10^5$ calls/s at $T = 1$ to 2.3–$2.5 \cdot 10^6$ at $T = 8$ ($\times 7.0$ for $c = 40$, $\times 8.1$ for $c = 48$). The median wall time for $c = 40$ fell from 8.7 s to 4.0, 2.4 and 1.1 s for $T = 2, 4, 8$, and for $c = 48$ from 149 s to 62, 36 and 23 s. Wall-time speedups vary more than throughput, because the number of calls per run varies (from 0.3 to $11 \times 2^{c/2}$). The median cost was 2.0–$3.5 \cdot 2^{c/2}$ calls; the excess over $\sqrt{\pi} \cdot 2^{c/2}$ comes from the bounded, overwriting table of `pcs_mt` (Remark 7), so below we use the complete-table constant $\sqrt{\pi}$. On the full Keccak-$f[1600]$ with $T = 8$ (8 runs each), the median wall time was 0.20 s for $c = 40$ and 3.0 s for $c = 48$, at $2.3 \cdot 2^{c/2}$ calls and about $1.55 \cdot 10^6$ calls per second per core. All 144 bridges pass an independent checker built on the XKCP reference code. At this rate, 128 cores would need about 35 seconds for $c = 64$ and about 2.5 hours for $c = 80$ ($\sqrt{\pi} \cdot 2^{40} = 2^{40.83}$ calls), with a distinguished-point table of about 12 GiB. This projection assumes random-function behaviour and linear scaling to 128 cores, which we have not yet measured. No time–memory trade-off beats the proven $\Theta(2^{c/2})$ bound [12, 52].

Incremental distinguished-point search. The $c = 80$ and $c = 88$ campaigns use a complete-table, incremental variant of the distinguished-point claw search (`09_distributed_search/`), stated as Algorithm 3. A single campaign walk key K fixes the walk map; the table of distinguished endpoints only grows, so a search that has already found k bridges can be continued to the $(k+1)$-st without repeating the stored work (the target k may be raised between cycles). The expected cost $E[T_{k+1}]$ is the heuristic of Theorem 4; Lemma 2 records what the procedure itself guarantees. A restart reuses the same K and the current T and B. Workers keep already-written complete records and truncate a partial last line; walks that had not yet reached a distinguished point, or had not yet been written, are not retried, because new walks draw new seeds. Interrupted rewalks of already-stored collisions stay pending and are retried. Same-side merges and Robin Hood walks (a walk that starts on an earlier trail) are recorded but not reported as bridges. The implementation also supports longer block shapes $(k_L, k_R > 1)$; the campaigns of this paper use $k_L = k_R = 1$, as above.

Lemma 2 (Incremental search)**.** *Let Algorithm 3 be run at a fixed capacity x, and let B be the list of bridges it reports.*

(i) Soundness. *Every entry of B is a bridge on x in the sense of Definition 4 (and hence a 2-block bridge at capacity x in the sense of Definition 2 and Theorem 1).*

(ii) Distinctness. *The first halves P_1 of the entries of B are pairwise distinct, which is the condition on the a_i in Definition 4 required by Theorem 4. Consequently the pairs (P_1, P_3) are pairwise distinct as well.*

(iii) Resumption. *The walk map g is a deterministic function of K. The table T only grows. Consequently, a resumed run with the same K and the complete table T detects every claw among the walks already stored in T that a single uninterrupted run with the same stored walks would detect. It may miss a claw that would have been found only with a walk that was still in flight at a restart, or whose last incomplete record was truncated, because those starts are not retried. Pending rewalks of stored collisions are not lost.*

The expected cost $E[T_{k+1}]$ remains the heuristic of Theorem 4.

Algorithm 3 Incremental distinguished-point bridge search

Require: fixed capacity x, walk key K, parameters c, d, p, target $k \geq 1$

from K, derive a keyed injective embedding e_K and a selector s_K (forward with probability p)
$g(z) \leftarrow \operatorname{cap} f(e_K(z)\|x)$ if $s_K(z) = 0$, else $\operatorname{cap} f^{-1}(e_K(z)\|x)$
$T \leftarrow$ persistent table of distinguished endpoints ▷ starts empty; never overwritten
$B \leftarrow$ list of reported bridges ▷ starts empty, or is restored on resume
while $|B| < k$ **do**
    draw a fresh $\max(2c, 256)$-bit seed; let z_0 be the c-bit start derived from the seed
    walk $z_{i+1} = g(z_i)$ until z_L has its low d bits equal to zero (abandon the trail if it is too long)
    store $(z_L, z_0, L, \text{seed})$ in T
    if T already has a walk ending at z_L **then**
        re-walk both trails to their merge; let u, v be the predecessors of the merge
        if $s_K(u) \neq s_K(v)$ and the triple $(z_L, y, \{u, v\})$ is new **then** ▷ $y = \operatorname{cap} f(e_K(u_{\text{fwd}})\|x)$
            $P_1 \leftarrow e_K(u_{\text{fwd}})$, $P_3 \leftarrow e_K(v_{\text{bwd}})$, $M \leftarrow \operatorname{rate} f(P_1\|x) \oplus \operatorname{rate} f^{-1}(P_3\|x)$
            if $f\big(f(P_1\|x) \oplus (M\|0^c)\big) = P_3\|x$ **then**
                if P_1 equals the first half of some entry of B **then**
                    record the claw as a duplicate first half (not counted in B)
                else
                    append (P_1, M, P_3) to B
                end if
            end if
        end if
    end if
end while
return B

Proof. For (i): a candidate is appended to B only if the two-call check $f(f(P_1\|x) \oplus (M\|0^c)) = P_3\|x$ succeeds (in the implementation the same identity is checked twice, in C and in Python). Writing $f(P_1\|x) = (v\|y)$ and $b = v \oplus M$, this is $f(b\|y) = (P_3\|x)$, which is Definition 4 with $a = P_1$ and $w = P_3$. The implication does not depend on how the candidate was produced, nor on any number of resumptions. It is machine-checked in Lean 4 for an arbitrary function f in `IncrementalSearch.lean`. For (ii): a candidate is appended to B only if its first half P_1 is not already the first half of an entry of B (a verified claw with a repeated P_1 is stored as a duplicate first half but is not counted in B and is not used for k^L messages). Hence the first halves of B are pairwise distinct. This is machine-checked in Lean 4 as `collect_firstHalf_nodup`; `collect_to_kL` supplies both hypotheses of Theorem 4 (every entry is a bridge on x, and the $a_i = P_1$ are pairwise distinct). For (iii): g is determined by K and the campaign parameters, and every complete distinguished-point record is kept (the table is never overwritten). Any claw whose both walks are already in T is therefore visible after a restart, and an unfinished rewalk stays pending. A walk that had not been written as a complete record is dropped (partial last lines are truncated) and is not repeated, since subsequent starts are drawn afresh. The cost estimate of Theorem 4 is the usual random-mapping heuristic and is not formalized. □

2.7 A public Keccak capacity-bridge challenge

We announce a public challenge on the full 24-round Keccak-$f[1600]$. A submission is a capacity c and three blocks P_1, M, P_3 of $r = 1600 - c$ bits such that, from the all-zero state, absorbing P_1, applying f, absorbing M, applying f and XORing P_3 into the rate leaves the entire 1600-bit

state equal to 0^{1600}, i.e. a bridge at the capacity with all zero bits (Definition 2): equivalently, $f\big(f(P_1\|0^c)\oplus(M\|0^c)\big) = (P_3\|0^c)$. We require $c \equiv 0 \pmod 8$, $8 \le c \le 1592$, so that the capacity is byte-aligned and the encoding is unambiguous; this interval is only the encoding range of the record table, not a claim that the search cost below applies at every such c. The metric is the largest c; for equal c the earliest public submission ranks first. A deterministic, dependency-free verifier (a pure Python Keccak-$f[1600]$ that checks itself against `hashlib.sha3_256`), the submission format, the rules and the record table are in the artifact folder `challenge/`. The $c = 80$ bridge of Remark 7 is the first solved instance. The expected generic cost is about $\sqrt{\pi}\cdot 2^{c/2} = 2^{c/2+0.83}$ calls of f and f^{-1}, in the parameter regime in which the search applies. The injective embedding of c-bit labels into r-bit rate blocks used by Algorithm 3 requires $r \ge c$, and even a two-list search with one-block halves needs enough rate words on each side for birthday-scale lists (at $c = 1592$, $r = 8$, each side has only 256 possible rate words). Within that regime the displayed cost is a heuristic for an ideal permutation, not a proven bound, although it agrees with our measurements on Keccak-$f[1600]$ and on a true random permutation (Remark 7). Since release 1.7 the verifier accepts every integer c with a bit-level split of the state (the rate is bits $0, \dots, r-1$ of the state read as one little-endian integer); such bridges, for example the lucky upgrades of Remark 12, are verified and reported with their effective capacity, while the record table keeps byte-aligned capacities.

2.7.1 One-sided forgery challenges

A second kind of challenge publishes only the forward half of a bridge that we minted. For a bridge on x in the three-block form $f\big(f(P_1\|x_{\text{in}})\oplus(M\|0^c)\big) = (P_3\|x_{\text{out}})$ we publish c, x_{in}, x_{out} and P_1, i.e. the public key (x, a) of Protocol 4, and the task is to find *any* completion (M, P_3). Minting costs about $2^{c/2+0.83}$ calls, while completing a given half is a fixed-target problem of heuristic generic cost about 2^c calls; about $2^{1600-2c}$ completions exist. Bridges 3 and 4 of our $c = 88$ campaign (Section 3.4.2) are published only in this form, as the three challenges of Table 2. Bridge 3 has effective capacity 92 (Remark 12), so it is also posed as a generalized challenge at $c = 92$, from $(\alpha\|x)$ to $(\omega\|x)$ with $\alpha \ne \omega$; by Lemma 5, a completion at $c = 92$ is also a completion of the $c = 88$ challenge. Publishing x_{out} at $c = 92$ reveals the four bits ω of P_3, which does not help a forger, since any completion is accepted.

Table 2: One-sided forgery challenges on the full Keccak-$f[1600]$, $x =$ `cef2e4428344e8e8ad398f`. Capacities are hexadecimal little-endian integers, so at $c = 92$ the four new low-order bits α and ω are the last hex digit. P_1 (189 bytes) is abbreviated; the full values are in Appendix C and in the artifact (`challenge/one_sided/` in Zenodo [33]).

Challenge	c	$x_{\text{in}} \to x_{\text{out}}$	P_1	c_{eff} (gen./loop)	cost
c88-B3	88	$x \to x$	`98061a11...f02d924b`	92 / 88	2^{88}
c92-B3	92	`...398f4` → `...398fa`	`98061a11...f02d920b`	92 / –	2^{92}
c88-B4	88	$x \to x$	`21077461...22e16d73`	89 / 88	2^{88}

Salted SHA3-256 commitments:
c88-B3, c92-B3: `135259d6611d2be43a8b83fda6cad933a38ef54dd6ebbed6b343262cfe50a545`
c88-B4: `2593472e5e2f119144820588a33dd9583809a2990bd8bee88878e71ade5fc52d`

Commitments and rules. For each of bridges 3 and 4 we publish

$$\mathsf{Com} = \text{SHA3-256}(\texttt{"BEH-onesided-v1"} \,\|\, \texttt{0x00} \,\|\, s \,\|\, \mathsf{canon}),$$

where s is a fresh secret 32-byte salt and canon is the canonical JSON encoding of the full bridge (c, the number of rounds, x_{in}, x_{out}, P_1, M, P_3 and nothing else). A submission is a pair (M, P_3)

for one challenge, in the two-call form only; the public verifier accepts *every* valid completion, and the earliest publicly timestamped valid submission solves the challenge. The challenges c88-B3 and c92-B3 share one salted commitment. Any valid completion of either view closes both views at the same public timestamp; the record states which level the completion actually passed ($c = 88$ or $c = 92$). A completion that passes at $c = 92$ receives credit for both c88-B3 and c92-B3, since by Lemma 5 it is also a completion at $c = 88$. Only after that closing is the common commitment opened. Submissions to either B3 view after the closing timestamp are not accepted. The challenge c88-B4 has its own commitment and is independent. Among the estimated $2^{1600-2c}$ completions at a given level, 2^{-1424} at $c = 88$ is the probability of hitting our own completion in a uniform draw; an honest solver's output distribution is unspecified. A submission equal to the committed bridge (which we can check privately with the salt) is treated as evidence that our copy leaked, not as a solution, and we then open the commitment so that anyone can check the claim. After a valid submission, or when a challenge is retired, we publish s and canon.

Trust model. The challenges assume that our private copies of the bridges stay secret. During the search the walk key and the distinguished-point table, from which the bridges can be recomputed cheaply, were stored on university servers and on a cloud assistant machine. The copies on the university servers were deleted on 12.11.2026, before publication; the remaining copies stay under the author's control. We do not claim that deleted data is unrecoverable (snapshots, backups and remapped storage blocks may keep it), which is why a submission equal to the committed bridge counts as a compromise. A zero-knowledge proof of the relation of Protocol 4 alone would show possession of some valid completion without opening the commitment; establishing that it is the specific committed completion also requires checking the commitment in the proof relation. Such a proof does not protect against such a leak.

3 Bridge multicollisions: k^L for every length

Theorem 3 reuses one inner collision and one connector in every block slot. Here we use *several* closed two-block paths at the same capacity value x and let the message choose, in every slot, which of them to take. The result is a k^L-fold multicollision for every length parameter L, with all messages of the same length, at a search cost that depends on k but not on L.

3.1 Setup

Let f be a permutation of $\{0,1\}^{r+c}$, and write a state as $(R\|C)$ with $R \in \{0,1\}^r$, $C \in \{0,1\}^c$. Absorbing a block $m \in \{0,1\}^r$ maps $(R\|C) \mapsto f((R \oplus m)\|C)$. Fix a capacity value $x \in \{0,1\}^c$ (in general $x \neq 0^c$).

Definition 4 (Bridge on x)**.** A *bridge on* x is a pair $(a, b) \in (\{0,1\}^r)^2$ such that

$$f(a\|x) = (v\|y), \qquad f(b\|y) = (w\|x)$$

for some $v, w \in \{0,1\}^r$ and $y \in \{0,1\}^c$. Bridges $(a_1, b_1), \ldots, (a_k, b_k)$ on x are *distinct* if $a_1, \ldots, a_k$ are pairwise distinct.

This is the two-block MITM loop of Theorem 1 written with its two half-states: the forward half a and the backward half b meet in the capacity y. Only the first rate inputs a_i are required to be distinct; the middle capacities y_i may coincide.

Lemma 3 (Free rate absorption)**.** *Let* (a, b) *be a bridge on* x *with* $f(a\|x) = (v\|y)$ *and* $f(b\|y) = (w\|x)$*. For every* $R \in \{0,1\}^r$ *the two blocks* $(R \oplus a,\ v \oplus b)$ *lead the sponge from* $(R\|x)$ *to* $(w\|x)$*.*

Proof. Absorbing $R \oplus a$ in $(R\|x)$ gives $f((R \oplus R \oplus a)\|x) = f(a\|x) = (v\|y)$. Absorbing $v \oplus b$ in $(v\|y)$ gives $f(b\|y) = (w\|x)$. □

The end state $(w\|x)$ does not depend on R: the incoming rate is absorbed for free.

Lemma 4 (Composition)**.** *Let (a_i, b_i), $i \in [k]$, be distinct bridges on x with $f(a_i\|x) = (v_i\|y_i)$ and $f(b_i\|y_i) = (w_i\|x)$, and let $R_0 \in \{0,1\}^r$. For a word $\sigma = (\sigma_1, \ldots, \sigma_L) \in [k]^L$ put $R_0^\sigma = R_0$, $R_j^\sigma = w_{\sigma_j}$, and*

$$M_\sigma = \big(R_0^\sigma \oplus a_{\sigma_1},\ v_{\sigma_1} \oplus b_{\sigma_1},\ R_1^\sigma \oplus a_{\sigma_2},\ v_{\sigma_2} \oplus b_{\sigma_2},\ \ldots,\ R_{L-1}^\sigma \oplus a_{\sigma_L},\ v_{\sigma_L} \oplus b_{\sigma_L}\big).$$

Then M_σ has $2L$ blocks and leads from $(R_0\|x)$ to $(R_L^\sigma\|x)$, where $R_L^\sigma = w_{\sigma_L}$ for $L \geq 1$ and $R_0^\sigma = R_0$ for the empty word ($L = 0$, M_σ empty), passing through capacity x after every second block. If $\sigma \neq \sigma'$ then $M_\sigma \neq M_{\sigma'}$.

Proof. By induction on j using Lemma 3: after the first $2j$ blocks the state is $(R_j^\sigma\|x)$, and the next two blocks lead to $(w_{\sigma_{j+1}}\|x) = (R_{j+1}^\sigma\|x)$. For injectivity, let j be the first index with $\sigma_j \neq \sigma'_j$. Then σ and σ' agree before j, so $R_{j-1}^\sigma = R_{j-1}^{\sigma'}$, and block $2j-1$ is $R_{j-1}^\sigma \oplus a_{\sigma_j} \neq R_{j-1}^\sigma \oplus a_{\sigma'_j}$ because the a_i are distinct. □

3.2 Full-state multicollisions

Theorem 4 (k^L full-state multicollisions)**.** *Let (a_i, b_i), $i \in [k]$, be distinct bridges on x, let P be a block sequence that leads the sponge from its initial state to $(R_0\|x)$, and fix any $z \in \{0,1\}^r$. For $\sigma \in [k]^L$ let $t_\sigma = w_{\sigma_L} \oplus z$ (with w_{σ_L} replaced by R_0 if $L = 0$). Then the k^L messages*

$$P \,\|\, M_\sigma \,\|\, t_\sigma, \qquad \sigma \in [k]^L,$$

are pairwise distinct, all have $|P|+2L+1$ blocks ($|P|$ prefix blocks, the $2L$ blocks of M_σ, then the terminal block t_σ), and all lead to the same full state $f(z\|x)$. Consequently, for every common suffix S they have the same digest.

- *The statements above hold for every $k \geq 1$ and $L \geq 0$.*
- *Heuristically, for an ideal permutation and in the Poisson occurrence model of the proof, the distinguished-point claw search with forward share $0 < p < 1$ and $q = 2p(1-p)$ has the k-th useful claw* occur *after an expected*

$$E[T_k] \approx \frac{\Gamma(k+\frac{1}{2})}{\Gamma(k)}\sqrt{\frac{2 \cdot 2^c}{q}}$$

  calls of f and f^{-1}, independent of L. The equality is exact only within that model. It assumes that the distinguished-point table is retained without material loss, and that revisits, the rejection of duplicate claws and repeated first rate inputs have negligible effect, i.e. $E[T_k] \ll 2^c$ (so k is fixed, or $k = o(2^c)$, as c grows; at fixed r there are at most 2^r distinct first rate inputs, so $k \leq 2^r$ in any case). Detection and replay overhead of an implementation are separate and additive (Remark 11; cf. [73, §4.1]). In particular $E[T_1] = \sqrt{\pi/(4p(1-p))} \cdot 2^{c/2}$, and $E[T_k] = \sqrt{2k/q} \cdot 2^{c/2}(1 + O(1/k)) = \sqrt{4k} \cdot 2^{c/2}/\sqrt{2q} \cdot (1+O(1/k))$ for large k within the regime above; at $p = \frac{1}{2}$ ($q = \frac{1}{2}$) this is $E[T_1] = \sqrt{\pi} \cdot 2^{c/2}$ and $E[T_k] < \sqrt{4k} \cdot 2^{c/2}$.
- *Verifying the k bridges costs 2 calls of f per bridge; building one message costs no call of f.*

Proof. By Lemma 4, $P\|M_\sigma$ leads to $(w_{\sigma_L}\|x)$; absorbing t_σ gives $f((w_{\sigma_L} \oplus w_{\sigma_L} \oplus z)\|x) = f(z\|x)$. Distinctness follows from Lemma 4, since the messages have the same length and their middle parts differ. Equal full states give equal outputs after any common suffix, because the sponge is deterministic from its state onward. For the cost, the distinguished-point claw search of Section 2.6 at the fixed capacity x (forward map $a \mapsto \operatorname{cap} f(a\|x)$, backward map $b \mapsto \operatorname{cap} f^{-1}(b\|x)$) is continued after its first claw, as in Algorithm 3 (Lemma 2). After T evaluations whose trail information is retained (the implementation stores only the distinguished endpoints of the trails; a merge is detected once both trails reach their common distinguished point), the expected number of collisions of the walk map grows as $T^2/(2 \cdot 2^c)$, as for many collisions with distinguished points [73], and a collision is a useful claw (one forward and one backward point) with probability $q = 2p(1-p)$. Modelling (as an approximation) the useful claws as a Poisson process in $s = qT^2/(2 \cdot 2^c)$, the k-th claw arrives at $s_k \sim \text{Gamma}(k, 1)$, so $T_k = \sqrt{2 \cdot 2^c s_k/q}$ and $E[T_k] = \sqrt{2 \cdot 2^c/q}\, E[\sqrt{s_k}] = \sqrt{2 \cdot 2^c/q}\, \Gamma(k + \frac{1}{2})/\Gamma(k)$. For $k = 1$, $\Gamma(\frac{3}{2}) = \sqrt{\pi}/2$ gives $\sqrt{\pi 2^c/(2q)} = \sqrt{\pi/(4p(1-p))} \cdot 2^{c/2}$. By Wendel's inequality, $k/\sqrt{k + \frac{1}{2}} \le \Gamma(k+\frac{1}{2})/\Gamma(k) \le \sqrt{k}$, so $E[T_k] = \sqrt{2k/q} \cdot 2^{c/2}(1 + O(1/k))$, which is $\sqrt{4k} \cdot 2^{c/2}$ at $q = \frac{1}{2}$. This is the usual random-mapping heuristic, not a proof. Nothing in the search depends on L. □

Lemmas 3 and 4 and the combinatorial part of Theorem 4 (common full state $f(z\|x)$, pairwise distinctness, the count of k^L messages of length $|P| + 2L + 1$, and equal digests under any common suffix) are machine-checked in Lean 4 for an arbitrary function f in `KLMulticollision.lean`; the two-call soundness of the incremental collector and the pairwise distinctness of its first halves (Lemma 2) are machine-checked in `IncrementalSearch.lean`; the cost estimate is not formalized.

Remark 9 (Padding and prefix)**.** The block t_σ must be a full, freely chosen rate block, so it cannot carry padding. With SHA3 padding (`0x06...0x80`) and with Keccak padding (`0x01...0x80`), the padding and any domain bits are placed in the common suffix S and the final padded block(s); all messages have equal length, so the padded tail after the common state $f(z\|x)$ is identical for all of them, and the bit length before padding is $(|P|+2L+1)\,r+|S|$, with $|P|$ counted in blocks and $|S|$ in bits; it is byte-aligned when both r and $|S|$ are byte-aligned (bit-level padding remains valid otherwise). If S is empty and r is byte-aligned, the padding is one full extra block `06 00...00 80` (SHA3) or `01 00...00 80` (Keccak), absorbed after t_σ; this is the case in the $c = 80$ example of Section 3.4.1 (9 message blocks, then one common padding block). The prefix P must reach capacity x. If x is the initial capacity 0^c, then P may be empty. Otherwise P is a path to a given inner state (an *IV transition* $0^c \to x$), which has generic order $2^{c/2}$ [12, §5.5, pp. 40–42]; with the claw search above (forward walks from capacity 0^c, backward walks into x) a two-block transition $P = (a_0, v_0 \oplus b_0)$ to $(w_0\|x)$ has the same expected occurrence cost $E[T_1] = \sqrt{\pi/(4p(1-p))} \cdot 2^{c/2}$ as one balanced search in Theorem 4. Measured at $c = 80$, $p = \frac{1}{2}$: $2^{40.85}$ calls against $E[T_1] = 2^{40.83}$ (Section 3.4.1); herding [50] does not lower this cost, since a single fixed target is reached once and the diamond structure would add, not save, work.

Corollary 1. *Under the assumptions of Theorem 4:*

(i) $k = 1$*: the messages* $P\|M_{(1,\ldots,1)}\|t$ *form an* expandable message *reaching the state* $f(z\|x)$ *with every length* $|P|+2j+1$*,* $j \ge 0$ *(cf. the fixed-point expandable messages of [51, §3.2]).*

(ii) $k = 2$*: a* 2^L*-multicollision for every* L*, at a fixed heuristic cost of about* $(\Gamma(\frac{5}{2}) \cdot 2 + \sqrt{\pi}) \cdot 2^{c/2} \approx 4.43 \cdot 2^{c/2}$ *at* $p = \frac{1}{2}$ *including the IV transition to* x*, against about* L *inner-collision searches of* $\approx \sqrt{\pi/2} \cdot 2^{c/2}$ *each for a binary Joux chain [46] (which needs no IV transition). In a permutation sponge such a chain uses about* $L + 1$ *blocks before padding (each choice block also absorbs the previous rate difference, and one final block aligns the rate), against* $|P| + 2L + 1$ *here.*

(iii) $k = 64$*: a* 2^{6L}*-multicollision for every* L*, at about* $2\,\Gamma(64.5)/\Gamma(64) \cdot 2^{c/2} \approx 15.97 \cdot 2^{c/2} \approx 2^{c/2+4.00}$ *for the bridges at* $p = \frac{1}{2}$ *(against* $\sqrt{4 \cdot 64} = 16$*).*

3.3 Related work and positioning

Paths to a given inner state cost about $2^{c/2}$ [12, §5.5, pp. 40–42], and Bertoni et al. already observe that one path from 0^c to 0^c gives an infinite class of such paths, at about $2^{(c+4)/2}$ [12, §5.10.2, p. 49]; our 2011 note obtains such a path with f and f^{-1} at a one-time cost of about $2 \cdot 2^{c/2}$ [34]. Multicollisions in iterated hashes are due to Joux [46], and expandable messages from fixed points to Kelsey and Schneier [51, §3.2]. For CubeHash, one path to the all-zero fixed state already gives multicollisions of arbitrary size at a cost independent of their size [3, §3]; the same construction is also described by Ferguson, Lucks and McKay [29, §1.3]. There the colliding messages differ in length.

Our contribution is modest: an explicit treatment of k capacity cycles at an arbitrary x, the k^L count of equal-length messages for every L, at a heuristic cost of about $\sqrt{4k} \cdot 2^{c/2}$ (at $p = \frac{1}{2}$; the Poisson-model estimate $E[T_k]$ of Theorem 4) independent of L; a binary Joux chain producing at least as many messages needs $\lceil L \log_2 k \rceil$ inner-collision searches of generic order $2^{c/2}$ each [46], [12, §5.10.1] (e.g. $k = 64$, $L = 10$: 2^{60} messages need 60 Joux stages, not 10), a construction that is correct under SHA3 and Keccak padding, and a full-round demonstration from the standard IV at $c = 80$ (Section 3.4). The k^L count with $k > 1$ cycles is what we consider new. It has no impact on SHA3-256: with $c = 512$ the cost is about 2^{256}. As a cryptanalytic tool it can feed Joux-style attacks on combiners such as $H_1 \| H_2$ and on cascades when one component is a sponge with a small capacity [46].

Remark 10 (One-time signatures from bridges)**.** Bridge halves suggest a Lamport-style challenge-response construction in which each signed bit reveals one half of a bridge (each half alone can be computed with f or f^{-1}). A proof of knowledge or of work would require a specified commitment and challenge protocol and a soundness analysis; we do not establish such a protocol here. In particular, without a prior binding commitment and an unpredictable challenge, a signer who knows the message in advance can compute only the halves it needs with ordinary evaluations, so the expected search cost $E[T_k]$ is not certified. XMSS or LMS [42, 60] with a proof-of-work stamp per leaf is a simpler alternative, so we do not claim this as a contribution.

A constructed haystack. Finding a needle in a haystack is usually the hard search; here we construct the haystack itself. Theorem 4 gives, for every L, an explicit set of k^L messages of equal length $|P| + 2L + 1$ blocks, all reaching the same full state $f(z\|x)$ and hence the same digest, and anyone holding the bridges can enumerate this set or sample from it uniformly at will. The asymmetry is that building the set costs about $E[T_k] \approx \sqrt{4k} \cdot 2^{c/2}$ (heuristically, at $p = \frac{1}{2}$, plus the IV transition), after which each member requires no further search by its builder (writing it out still costs time linear in L), whereas without the bridges finding even one member costs a generic search on the capacity. The set alone gives no hiding or unlinkability, since a sampled member reveals the bridge blocks it uses. We invite the community to find uses for such structured, samplable equal-digest sets, for example as a tool for multicollision-based attacks on combiners that include a small-capacity sponge, or as a building block for new primitives.

3.4 Demonstrations

3.4.1 A full standard-IV example at $c = 80$

We instantiate Theorem 4 on the full 24-round Keccak-$f[1600]$ with $r = 1520$, $c = 80$, starting from the standard all-zero IV of a SHA3-256-structured sponge with reduced capacity, with

Table 3: The $c = 80$ standard-IV 3^3-multicollision on the full Keccak-$f[1600]$ ($r = 1520$, $d = 20$, $p = \frac{1}{2}$). Calls are fleet-wide calls of f and f^{-1} at detection; expectations from Theorem 4.

Object	capacities	calls (observed)	$E[\cdot]$	intermediate capacity y
Transition T	$0^{80} \to x$	$2^{40.85}$	$2^{40.83}$	`c1859fd5ffea04084956`
Bridge B_3 (1st found)	$x \to x$	$2^{41.45}$	$2^{40.83}$	`7b1a26bd4bd1605457e6`
Bridge B_1 (2nd found)	$x \to x$	$2^{41.83}$	$2^{41.41}$	`ee95a4152637611daf26`
Bridge B_2 (3rd found)	$x \to x$	$2^{42.10}$	$2^{41.73}$	`851b3543804e3f48702a`
Total (A then B)		$2^{42.61}$	$2^{42.35}$	
$x =$ `73e10d5cd30ae96a5434`; 27 messages of 9 blocks (1710 bytes) + 1 SHA3 padding block (2 transition, 6 bridge, 1 final block); common SHA3-256-structured digest `14a4d530...040f6fc7`				

$k = 3$ and $L = 3$. The capacity $x =$ `73e10d5cd30ae96a5434` was drawn at random (nonzero) before the searches. Both searches used the distributed distinguished-point claw search (code release v1.6, Appendix A) with $d = 20$, $p = \frac{1}{2}$ ($q = \frac{1}{2}$) and a complete distinguished-point table, run one after the other on the same machines (about $2^{28.77}$ calls/s combined).

- *Run A (bridges on x).* Forward and backward walks both at capacity x. The three bridges were detected at $2^{41.45}$, $2^{41.83}$ and $2^{42.10}$ calls (4,512,718 = $2^{22.11}$ distinguished points, 3.0 h); the Gamma law of Theorem 4 predicts $E[T_1], E[T_2], E[T_3] = 2^{40.83}, 2^{41.41}, 2^{41.73}$, so the third bridge came at about $1.29 \times E[T_3]$.
- *Run B (IV transition $0^{80} \to x$).* Forward walks from capacity 0^{80}, backward walks into x. The transition was found after $2^{40.85}$ calls ($2^{20.85}$ distinguished points, 1.3 h), against $E[T_1] = 2^{40.83}$.

The total is $2^{42.10} + 2^{40.85} = 2^{42.61}$ calls, against an expected $E[T_3] + E[T_1] = 2^{42.35}$. The transition blocks (P_1^T, M^T) take the IV $0^{1520}\|0^{80}$ to $(P_3^T\|x)$, so $P = (P_1^T, M^T)$ and $R_0 = P_3^T$. With $z = 0^r$ and an empty suffix, Theorem 4 gives $3^3 = 27$ messages of $2 + 6 + 1 = 9$ blocks, i.e. 1710 bytes, followed by one common SHA3 padding block (Remark 9). All 27 messages are pairwise distinct, reach the same full 1600-bit state and the same 256-bit digest

`14a4d530cdfdf137a60719c662f1a64f889d832583d512f0311adfa8040f6fc7`.

This was checked from the raw message bytes alone, without the bridge data, with two independent Keccak-$f[1600]$ implementations (a standalone Python Keccak and the XKCP `CompactFIPS202`, whose $c = 512$ instance matches `hashlib.sha3_256`), on two machines with different operating systems and CPU architectures. Table 3 summarises the example; the transition and bridge blocks are in Appendix B.

Distinct finds on disjoint trails. Tracing all 12 trails of the six collisions of Run A (three useful claws, one forward–forward and two backward–backward merges; the observed useful fraction 3/6 matches q, which is descriptive only for six events) gives 17,546,091 = $2^{24.06}$ distinct points. No trail of one claw shares a point with a trail of another claw, no claw point lies on another trail, and no seed or distinguished point is shared, so the three bridges are distinct, non-duplicated finds on disjoint recorded trails (the expected number of chance cross hits is about 2^{-33}). This does not by itself establish statistical independence of the finds or of their stopping times.

Remark 11 (Detection latency)**.** With W walks in flight, a claw is detected only when the later of its two walks reaches its distinguished point, and the fleet keeps working meanwhile. The delay is about $\Lambda \approx 2^d \cdot W$ calls, so the corrected expectation of the detected cost is $E[T_k] + \Lambda$. In Run A, two GPUs with about 2^{19} walk slots in total and $d = 20$ give $\Lambda \approx 2^{39.0}$, about $0.28\,E[T_1]$;

Table 4: The four $c = 88$ bridges on $x =$ `cef2e4428344e8e8ad398f` ($r = 1512$, $d = 22$, $p = 0.66$). Calls are fleet-wide calls of f and f^{-1} at detection; $E[T_k]$ from Theorem 4; $s_k = qT_k^2/(2 \cdot 2^c)$ and $\Pr[S_k \le s_k]$ for $S_k \sim \text{Gamma}(k)$; c_{eff} from Remark 12. The claw label is the search's internal name of P_1.

k	claw label	found	calls	$E[T_k]$	ratio	$\Pr[S_k \le s_k]$	c_{eff}	published
1	`63db8606...`	01.11.2026	$2^{44.79}$	$2^{44.90}$	0.92	0.49	89 / 89	full
2	`1e00f374...`	01.11.2026	$2^{45.16}$	$2^{45.49}$	0.80	0.31	88 / 88	full
3	`f70a0908...`	09.11.2026	$2^{45.29}$	$2^{45.81}$	0.70	0.15	92 / 88	one-sided
4	`990e297d...`	10.11.2026	$2^{45.82}$	$2^{46.03}$	0.86	0.31	89 / 88	one-sided

indeed the fleet counter was $2^{42.10}$ at the third find, while the completed walks summed to $2^{41.93}$. The corrected expectations are $2^{41.18}$, $2^{41.66}$ and $2^{41.93}$, and the observed values are 1.20, 1.13 and 1.12 times these. Under the constant-latency approximation a common mean delay cancels in the expected increments between bridges (realized delays vary and can even reorder detections); the observed increments $+2^{39.75}$ and $+2^{39.56}$ are compatible with the predicted $+2^{39.83}$ and $+2^{39.42}$, without being a separate test of the distribution. Over the four first bridges we found at $c = 80$ (Remark 7, a second run, a generalized-capacity run and Run A), Fisher's combination of the per-run tail probabilities gives $p = 0.15$ with the latency correction (0.094 without), so the lateness is consistent with chance. These statistics (the fleet counters, the trail-overlap audit and the Fisher combination) are unreplicated observations from our run records; the release (since v1.6.1) contains the run counters and settings, the overlap audit (its inputs, script and output; re-tracing the walks needs the unpublished walk key) and the Fisher script with its selection rule, tail definition and latency assumption (`bridges/c80_kL27/run_records/`). For CPU-only runs and for larger d/c settings with fewer slots the effect is small (e.g. $\Lambda/E[T_1] \approx 0.07$ in our $c = 88$ run).

3.4.2 Four bridges at a generalized capacity, $c = 88$

At $c = 88$ ($r = 1512$, $d = 22$, $p = 0.66$, so $q = 0.4488$ and $E[T_1] = 2^{44.90}$) we ran the distributed search on the nonzero capacity $x =$ `cef2e4428344e8e8ad398f` in three launches (31.10.2026–01.11.2026 for two bridges, then 09.11.2026 and 09.11.2026–10.11.2026 for one more bridge each), always resuming with the same walk key and the complete distinguished-point table (Algorithm 3, Lemma 2), so that the call counts are cumulative. The four claw labels of Table 4 are pairwise distinct, hence so are the first halves $P_1 = e_K(u)$ (e_K is injective), and the run satisfies the distinct-bridges hypothesis of Theorem 4. The four bridges were detected after $2^{44.79}$, $2^{45.16}$, $2^{45.29}$ and $2^{45.82}$ calls (Table 4). Bridges 1 and 2 are public; bridges 3 and 4 are published only by their forward halves, as the one-sided forgery challenges of Section 2.7.1.

Incremental search and the Gamma law. In the Poisson model of Theorem 4 the increments $s_k - s_{k-1}$ are independent and exponential with mean 1. The observed increments are 0.67, 0.45, 0.22 and 1.46; Fisher's combination of their upper tails gives $p = 0.69$ ($\chi^2 = 5.60$ with 8 degrees of freedom), and the Kolmogorov–Smirnov distance to Exp(1) is 0.26 (5% critical value 0.62 for four values). All four bridges came somewhat earlier than $E[T_k]$ (ratios 0.70–0.92, Figure 3), within the central 80% of the predicted distribution of T_k in every case. Four events do not test the model sharply, and the counts are not corrected for detection latency (Remark 11) or for walks lost at the restarts, both of which would make them larger. The script `bridges/c88/run_records/gamma_c88.py` reproduces these numbers.

Messages. With bridges 1 and 2 and $L = 3$ we assembled 8 distinct 1323-byte messages (7 blocks, then a common SHA3 padding block) that start from the state $0^r \| x$ (empty prefix) and

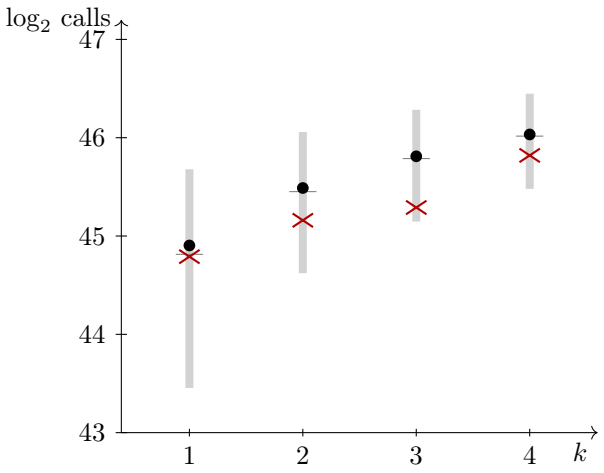

Figure 3: Calls to the k-th $c = 88$ bridge ($\log_2$). Grey bar: 10% to 90% quantiles of T_k under the Gamma law of Theorem 4 (tick: median); dot: $E[T_k]$; cross: observed.

reach the same 1600-bit state and digest

`0bf519ab4634d0b5460831292b162652752234e76893ad617c664eb14ee97119`.

The IV transition $0^{88} \to x$ *was not computed* (it would cost about $2^{44.83}$ calls at $p = \frac{1}{2}$), so this is a generalized-capacity example only; Section 3.4.1 is the example with a full standard-IV transition. With all four bridges, Theorem 4 gives $4^3 = 64$ messages of the same length with the same final state $f(0^r\|x)$, hence the same digest; we assembled and checked them privately. Only the 8 messages that use bridges 1 and 2 can be published, since each of the other 56 reveals the blocks of bridge 3 or 4. They are determined by the committed bridges and can be checked once the commitments are opened. The public bridges, the 8 messages and the run records are in release v1.7 (`bridges/c88/`); the message verifier checks them from the start state $0^r\|x$ only when that state is given explicitly (option `-start-cap`).

3.5 Downward closure and lucky upgrades

The split of a 1600-bit state into rate and capacity is a convention of the sponge, not of the permutation. Moving the split point therefore changes the capacity at which a bridge is read, but not the bridge itself. We use the generalized form, in which the start and end capacities may differ. For integers $r, c \geq 0$ with $r + c = 1600$, a *generalized bridge at capacity* c *from* x_{in} *to* x_{out} is a pair $(a, b) \in (\{0,1\}^r)^2$ with

$$f(a\|x_{\text{in}}) = (v\|y), \qquad f(b\|y) = (w\|x_{\text{out}}) \tag{2}$$

for some $v, w \in \{0,1\}^r$ and $y \in \{0,1\}^c$, where $x_{\text{in}}, x_{\text{out}} \in \{0,1\}^c$. For $x_{\text{in}} = x_{\text{out}} = x$ this is a bridge on x (Definition 4); in general it is a two-block connector from x_{in} to x_{out} (Definition 1): from any state $(R\|x_{\text{in}})$ the blocks $(R \oplus a,\ v \oplus b)$ lead to $(w\|x_{\text{out}})$, as in Lemma 3. In the three-block form used by our verifier, $P_1 = a$, $M = v \oplus b$ and $P_3 = w$, so that $f\big(f(P_1\|x_{\text{in}}) \oplus (M\|0^c)\big) = (P_3\|x_{\text{out}})$. For $c' \leq c$ and $z \in \{0,1\}^c$ we write $z = (\hat{z}\|z')$ with $\hat{z} \in \{0,1\}^{c-c'}$ and $z' \in \{0,1\}^{c'}$; z' is the c'*-bit truncation* of z, i.e. its last c' bits, which stay in the capacity when the capacity shrinks to c'.

Lemma 5 (Downward closure)**.** *Let* (a, b) *be a generalized bridge at capacity* $c_{\max}$ *from* x_{in} *to* x_{out}*, as in* (2)*, and let* $0 \leq c \leq c_{\max}$*,* $t = c_{\max} - c$*. Write* $x_{\text{in}} = (\alpha\|x'_{\text{in}})$*,* $y = (\eta\|y')$ *and* $x_{\text{out}} = (\omega\|x'_{\text{out}})$ *with* $\alpha, \eta, \omega \in \{0,1\}^t$*. Then*

$$(a', b') = (a\|\alpha,\ b\|\eta) \in (\{0,1\}^{r+t})^2$$

is a generalized bridge at capacity c from x'_{in} to x'_{out}, with middle capacity y', $v' = v\|\eta$ and $w' = w\|\omega$. Its three-block form is $P'_1 = P_1\|\alpha$, $M' = M\|0^t$, $P'_3 = P_3\|\omega$. In particular, a bridge on x at capacity $c_{\max}$ is a bridge on the c-bit truncation x' of x at every capacity $c \le c_{\max}$.

Proof. As 1600-bit strings, $(a'\|x'_{\text{in}}) = (a\|\alpha\|x'_{\text{in}}) = (a\|x_{\text{in}})$; only the position of the rate/capacity split differs. Hence $f(a'\|x'_{\text{in}}) = f(a\|x_{\text{in}}) = (v\|y) = (v\|\eta\|y') = (v'\|y')$, and the middle capacity at capacity c is y'. In the same way $(b'\|y') = (b\|\eta\|y') = (b\|y)$, so $f(b'\|y') = f(b\|y) = (w\|x_{\text{out}}) = (w\|\omega\|x'_{\text{out}}) = (w'\|x'_{\text{out}})$. This is (2) at capacity c for (a', b'). Finally $M' = v' \oplus b' = (v \oplus b)\|(\eta \oplus \eta) = M\|0^t$, $P'_1 = a'$ and $P'_3 = w'$. If $x_{\text{in}} = x_{\text{out}} = x$, then $\alpha = \omega$ and $x'_{\text{in}} = x'_{\text{out}} = x'$. □

The t capacity bits that are dropped become rate bits: the top bits of x_{in} enter P'_1, the top bits of the middle capacity y enter v' and b' (and cancel in M'), and the top bits of x_{out} enter P'_3. No new search is involved; the lemma holds for every permutation f.

The converse: lucky upgrades. Read backwards, the proof also says when a bridge found at capacity c is a bridge at a larger capacity. Let (a', b') be a generalized bridge at capacity c and let $t \ge 1$. If the last t bits of $M' = v' \oplus b'$ are zero, then removing these t bits from a' and b' gives a generalized bridge at capacity $c + t$ from $(\alpha\|x'_{\text{in}})$ to $(\omega\|x'_{\text{out}})$, where α and ω are the last t bits of P'_1 and P'_3; this is Lemma 5 applied in reverse, since the two middle states $f(a'\|x'_{\text{in}})$ and $f^{-1}(P'_3\|x'_{\text{out}})$ then agree on $c + t$ capacity bits. If moreover $\alpha = \omega$, it is again a bridge on one capacity value, $(\alpha\|x')$, as needed for Theorem 4.

Remark 12 (Effective capacity)**.** For a bridge found at capacity c, let the *effective capacity* c_{eff} be the largest $c + t$ for which the upgrade above applies, i.e. c plus the number of consecutive zero bits of M counted from its last bit, and let $c_{\text{eff}}^{\text{loop}} \le c_{\text{eff}}$ be the largest $c + t$ for which in addition the last t bits of P_1 and P_3 agree. The search fixes only the c middle capacity bits, so for an ideal permutation the rate bits of M, P_1 and P_3 behave as uniform and independent, and per bridge, for $0 \le k \le r$,

$$\Pr[c_{\text{eff}} \ge c + k] = 2^{-k}, \qquad \Pr[c_{\text{eff}}^{\text{loop}} \ge c + k] = 2^{-2k},$$

while both probabilities are 0 for $k > r$ (a rate block has only r bits). The second display presumes an initial loop, i.e. a bridge already on a single capacity value at capacity c. The finite means are therefore $E[c_{\text{eff}} - c] = 1 - 2^{-r}$ and $E[c_{\text{eff}}^{\text{loop}} - c] = (1 - 4^{-r})/3$ (the values 1 and $\frac{1}{3}$ are the $r \to \infty$ limits), and the best of N independent bridges has $c_{\text{eff}} \approx c + \log_2 N + 1$. These upgrades are free, and c_{eff} need not be a multiple of 8. For our four bridges at $c = 88$ on $x =$ `cef2e4428344e8e8ad398f` (Section 3.4.2) we observed $c_{\text{eff}} = 89$, 88, 92 and 89, and $c_{\text{eff}}^{\text{loop}} = 89$, 88, 88 and 88 (mean excesses 1.5 and 0.25). The third bridge is thus a 92-bit generalized bridge (from one 92-bit capacity to a different one), but only an 88-bit bridge on a single capacity value; it is posed as the challenge c92-B3 (Section 2.7.1). A value of at least 92 among four bridges has probability $1 - (15/16)^4 \approx 0.23$. Each value was confirmed by re-expressing the bridge at the larger capacity and running the bridge verifier on it. As a control, 2^{12} pairs of random Keccak-$f[1600]$ outputs agreed on each of the 40 rate bits next to the capacity with frequency 0.50 (per-bit agreement; joint agreement of all 40 bits would have probability 2^{-40}), with mean run length 1.02.

Corollary 2 (One-sided partials of one bridge)**.** *Let an owner hold a bridge (a, b) on x at capacity $c_{\max}$, publish (x, a) as the one-sided form of Protocol 4, and publish a salted commitment* Com *to b with a secret salt s, for example the commitment to the full bridge of Section 2.7.1. For every $c \le c_{\max}$, for example $c \in \{64, 72, 80, 88\}$ when $c_{\max} = 88$, the pair $(x', a\|\alpha)$, with $x = (\alpha\|x')$ and x' the c-bit truncation of x, is the one-sided form of a bridge on x' at capacity c, and its secret half $b\|\eta$ is determined by b and the public data, since η consists of the first*

$c_{\max} - c$ *bits of* $\operatorname{cap} f(a\|x)$. *Hence a single commitment backs the partials at all levels* $c \le c_{\max}$, *and opening* Com *(revealing* s *and* b*) proves all levels at once, with two permutation calls per level.*

Proof. By Lemma 5 with $x_{\text{in}} = x_{\text{out}} = x$, $(a\|\alpha,\ b\|\eta)$ is a bridge on x' at capacity c. Its forward half $a\|\alpha$ and x' are computed from (x, a), and its middle capacity bits η from $f(a\|x)$, so given b every level is checked with the calls $f(a\|x)$ and $f(b\|y)$ of Protocol 4. □

All partials carry the same information. Since $a\|\alpha$ together with x' is just (a, x) cut at a different place, the partial at the lowest level already reveals the whole forward half (x, a) at $c_{\max}$, and no partial reveals anything about b beyond the commitment. Announcing the partials one level at a time therefore raises the claimed level without revealing more, but it cannot hide the higher levels from someone who reads the lowest partial. Opening is all or nothing: revealing the secret half $b\|\eta$ at any level reveals b, and with it the bridge at $c_{\max}$.

Remark 13 (What a low-level partial does not prove)**.** The cost of an object at level c depends only on c, not on $c_{\max}$. Producing *some* bridge on the truncation x' at capacity c by the search of Theorem 1 costs about $\sqrt{\pi} \cdot 2^{c/2} = 2^{c/2+0.83}$ calls, e.g. $2^{32.83}$ at $c = 64$; so anyone can publish a competing one-sided partial on x' at level 64 together with a valid secret half at that cost, and a partial at level c is no evidence that its publisher holds a bridge at capacity $c_{\max}$. Completing a *given* published half at level c, i.e. finding a secret half for it, is the forging problem of Protocol 4 at capacity c and costs about 2^c calls (e.g. 2^{64} at $c = 64$), again independent of $c_{\max}$. Possession of the bridge at $c_{\max}$ is shown only by opening the commitment, by completing the one-sided form at $c_{\max}$, or by a zero-knowledge proof of the relation of Protocol 4 at capacity $c_{\max}$. Likewise, a forgery challenge posed at level c is a level-c problem, whatever the capacity at which the bridge behind it was minted.

Remark 14 (Challenge ladders)**.** A natural use of one-sided challenges is a ladder of increasing difficulty, for example $c = 64, 68, \ldots, 84$, where the next level is announced after the previous one is solved. Cutting all levels from one bridge does not work: by Corollary 2 every level publishes the same half, the first opening reveals the whole bridge, and a solution at level c is also a solution at level $c + 4$ with probability $1/16$ if the event is that four further middle bits match and the resulting output capacity is free, and with probability $1/256$ if in addition a prescribed four-bit endpoint extension must match (under the independence model of Remark 12: of 256 pairs of independent four-bit values for these two conditions, 16 meet the first and one meets both). A ladder that publishes a prescribed x_{in} and x_{out} at the next level therefore accepts the second, joint event. A ladder therefore needs an independently minted bridge per level, each with its own capacity value, published half and salt; minting the levels $64, 68, \ldots, 84$ costs about $2^{43.2}$ calls in total, less than one bridge at $c = 88$. Even then a strong solver may skip levels, pauses between levels favour those who keep computing, and every submission must be checked as a completion, not against the owner's secret half. Levels near $c = 64$ are within practical reach of large computing resources (Table 5). We leave a full design, possibly with the zero-knowledge proofs of Section 4.2 for non-standard capacities, to future work.

4 Applications: Building Cryptographic Protocols with Keccak Bridges

Feasibility for capable entities. The protocols below ask an honest party to spend a tunable, expensive effort of about $2^{c/2}$ calls to mint a bridge, while a forger faces the 2^c wall. Table 5 puts this in physical terms, using three rates: a 128-core server running our code ($\approx 2^{27.7}$ Keccak-f[1600] calls/s, Section 2.6), an entity with 1% of the Bitcoin network, and the whole network ($\approx 2^{69.8}$ hashes/s [18]), under the same generous assumption as in Section 1.1

that one Keccak call costs as much as one Bitcoin hash. Minting is taken as $\sqrt{\pi} \cdot 2^{c/2}$ calls (the heuristic generic constant, which our complete-table distinguished-point search matches, Remark 7) and forging as 2^c. Under these throughput assumptions, which are not measure-

Table 5: Heuristic wall time to mint a MITM bridge ($\sqrt{\pi} \cdot 2^{c/2}$ calls) versus forging one (2^c calls).

	128-core server		1% of Bitcoin		Bitcoin network	
c	mint	forge	mint	forge	mint	forge
64	35 s	$2^{11.4}$ yr	< 1 ms	2 s	< 1 ms	0.02 s
80	2.5 h	$2^{27.4}$ yr	< 1 ms	33 h	< 1 ms	20 min
96	26 d	$2^{43.4}$ yr	< 1 ms	$2^{8.0}$ yr	< 1 ms	$2^{1.3}$ yr
128	$2^{12.2}$ yr	$2^{75.4}$ yr	3.2 s	$2^{40.0}$ yr	0.03 s	$2^{33.3}$ yr
160	$2^{28.2}$ yr	$2^{107.4}$ yr	2.4 d	$2^{72.0}$ yr	35 min	$2^{65.3}$ yr

ments, an entity at Bitcoin scale would mint at $c = 128$ in seconds and at $c = 160$ in days, while forging stays beyond 2^{33} years; a 128-core server (extrapolated from 8 cores) reaches $c \leq 96$. The capacity is thus a plausible tuning knob between "cheap for the minter" and "infeasible for everyone", subject to the caveats below. The same holds for the largest private AI clusters, which publicly report on the order of 10^6 GPUs [19]: assuming $3 \cdot 10^9$–10^{10} Keccak calls per GPU per second (an assumption, not a measurement), such a cluster could mint bridges up to $c \approx 134$ in a day and $c \approx 151$ in a year, while the 2^c wall limits forging to $c \lesssim 79$ even over a year. The rates are order-of-magnitude assumptions (no Keccak hardware exists at Bitcoin scale), the minting constant $\sqrt{\pi}$ is a heuristic (random-permutation model, matched by our measurements but not proven) and parallel efficiency is assumed linear.

All proposals below are design ideas; nothing is implemented or measured for these uses.

4.1 Proof-of-work with hardness-adaptive capacity

Motivation. Hash-based proof of work, as in Bitcoin, asks for a partial preimage: an input whose hash falls below a target. Its only knob is the target. The puzzle itself is fixed, and its cost is tied to no security parameter of the underlying primitive. Collision-type puzzles such as Equihash [17] add a birthday structure, but they need a separate generalised-birthday design and its own analysis. We want a puzzle built only from the standard Keccak permutation, whose expected cost is a generic bound of the primitive itself, namely the birthday cost $\approx 2^{c/2}$ of a capacity claw (Theorem 1). Its difficulty should be set by the capacity c, a parameter cryptographers already reason about. A solution should be checkable with two permutation calls and bound to the block header, so that work cannot be reused across blocks or epochs. Keccak bridges give exactly this. The rest of this subsection defines the puzzle, and Protocol 1 says how c follows the network's hash power over time.

The solution is an f_R/f_R^{-1} capacity claw at the capacity $z = \operatorname{cap} S_0$ of a header-derived start state (Theorem 1 with $x = z$; not 0^c)

$$S_0 = \mathrm{Expand}\big(H(\mathrm{prev} \,\|\, \mathrm{merkle} \,\|\, \mathrm{addr} \,\|\, \mathrm{epoch} \,\|\, \mathrm{ticket})\big).$$

Verification costs 2 reduced-round calls plus about 5 SHA3 calls. Difficulty: capacity c (coarse, $\times 2$ per 2 bits), a t-bit partial match ($\sqrt{2}$ steps), and a target T on $H(\mathrm{sol})$. A single claw search is not progress-free: time to success is Rayleigh (sd/mean 0.52, measured), so win probability scales like λ^2. Fixes: many small sub-puzzles or an Equihash-style generalised birthday [17]. Because the $R = 2$ slope is sub-generic, we recommend $R \geq 12$, ideally 24, and not R as a difficulty knob.

Epoch rule for the capacity upgrade. Protocol 1 fixes how the difficulty parameters $P_e = (c_e, t_e, T_e)$ change over time. Work is retargeted once per epoch, and the capacity is

Protocol 1 BridgePoW-Epoch: epoch rule for hardness-adaptive capacity (design only, not implemented).

Public constants. Round count R (we recommend $R \geq 12$, ideally 24); epoch length N blocks; target block time τ; clamp factor 4; floor $c_{\min}$ (even, chosen so that $2^{c_{\min}}$ forging is infeasible); hash $H = \text{SHA3-256}$. Expected work for parameters $P = (c, t, T)$ with even c, $t \in \{0, 1\}$, $T \leq 2^{256}$:

$$W(P) = \sqrt{\pi} \cdot 2^{(c+t)/2} \cdot 2^{256}/T.$$

Mining block h (epoch $e = \lfloor h/N \rfloor$, parameters P_e):

1. $S_0 \leftarrow \text{Expand}(H(\text{prev} \| \text{merkle} \| \text{addr} \| e \| \text{ticket}))$.
2. Let $z = \text{cap}\, S_0$. Find $\text{sol} = (a, w') \in \{0,1\}^r \times \{0,1\}^r$ such that $f_R(a \| z)$ and $f_R^{-1}(w' \| z)$ agree on the c_e capacity bits and on t_e further fixed rate bits (distinguished-point search, Section 2.6). Both endpoints carry the capacity z, which the verifier recomputes; a and w' are rate-only and use a fixed canonical encoding.
3. If $H(\text{sol}) < T_e$, publish the block with (P_e, sol); else go to step 1 with a new ticket (so that S_0 and z are recomputed).

Verifying block h**:** (i) P in the header equals P_e computed by the retarget rule below; (ii) recompute S_0 and $z = \text{cap}\, S_0$; (iii) check that sol is a pair of rate values in canonical encoding, and check the claw on $(a \| z)$ and $(w' \| z)$ with one f_R and one f_R^{-1} call; (iv) check $H(\text{sol}) < T_e$.

Retarget at each epoch boundary $e \to e + 1$:

1. $\rho \leftarrow \text{clamp}\big(\Delta_e/(N\tau),\, 1/4,\, 4\big)$, where Δ_e is the time taken by epoch e; $W^* \leftarrow W(P_e)/\rho$.
2. *Capacity upgrade:* $c_{e+1} \leftarrow \max\big(c_{\min},\ \text{largest even } c \text{ with } \sqrt{\pi} \cdot 2^{c/2} \leq W^*\big)$.
3. *Fine part:* $t_{e+1} \leftarrow 1$ if $\sqrt{\pi} \cdot 2^{(c_{e+1}+1)/2} \leq W^*$, else 0; $T_{e+1} \leftarrow \min\big(2^{256},\ 2^{256} \cdot \sqrt{\pi} \cdot 2^{(c_{e+1}+t_{e+1})/2}/W^*\big)$.
4. *Bounty override:* if epoch e contains a valid bounty claim (a ZK proof of a single-state bridge for c_e, Section 4), set $c_{\min} \leftarrow c_e + 2$ before step 2.

derived from the required work, never voted on. So c rises automatically as hash power grows, and precomputation cannot cross an epoch boundary, because the epoch number enters S_0.

$W(P)$ uses the heuristic distinguished-point constant $\sqrt{\pi}$ (Remark 7) and assumes the claw search behaves like a random mapping. It is the expected work of the restart strategy of step 3, which discards the search after each failed test $H(\text{sol}) < T_e$; the verifier cannot check that a miner does so. A miner that keeps one ticket, retains and enlarges both tables and tests every match against T_e needs, in the same idealized model, only $\Theta\big(2^{(c+t)/2}\sqrt{2^{256}/T}\big)$ permutation calls with memory of the same order, since two lists of n entries give about $n^2/2^{c+t}$ matches. Away from the capacity floor the retarget rule below keeps $T/2^{256} \in (2^{-1/2}, 1]$, so the two estimates differ by a factor of at most $2^{1/4} \approx 1.19$; for much smaller targets the difference would be larger. $W(P)$ is therefore an estimate for the specified strategy, not an established bound on optimal mining work. Each epoch's parameters are a deterministic function of the chain, so all honest nodes agree on them.

4.2 Bridges as zero-knowledge tokens

Motivation. Many systems need a scarce digital object whose scarcity comes from computation, not from a trusted issuer: proof-of-work coins, rate-limited identities, or certificates that someone invested a known amount of effort. Today such objects are usually a hash below a target. That proves the work but is public at once, it reveals the solution, and its cost has no structural link to a security parameter of the primitive. A Keccak bridge for public (R, c, tag) has the properties we want. It costs $\approx 2 \cdot 2^{c/2}$ to mint as a MITM bridge (Theorem 1), it is checked with a few permutation calls, and it can be kept secret behind a commitment and

Protocol 2 BridgeToken: minting, transfer and spending of bridge tokens (design only; the measured circuits of Tables 6 and 7 prove a related single-state relation, not this MITM relation, see below).

Public parameters. Round count R, capacity c (the denomination), ledger L, hash G = TurboSHAKE128, a ZK proof system with Fiat–Shamir context binding.

Mint (owner O with key pair (sk_O, pk_O)):

1. $S_0 \leftarrow \text{Expand}(G(\text{“bridge-v1”}\|R\|c\|\text{tag}\|pk_O))$, where tag is a fresh public value from L (for example a recent block hash), so no bridge can be precomputed.
2. Write $S_0 = (s\|z)$. Find a MITM bridge $B = L(s) = (s \oplus a, u \oplus w)$ at capacity z from rate s, i.e. exactly the two-block word of Theorem 1 with $x = z$ ($\approx 2 \cdot 2^{c/2}$ calls).
3. $C \leftarrow G(\text{“commit”}\|S_0\|B)$, $N \leftarrow G(\text{“null”}\|B)$.
4. Publish $(\text{tag}, C, N, \pi_{\text{mint}})$, where π_{mint} has public inputs $(R, c, \text{tag}, pk_O, C, N)$ and proves knowledge of a two-block word B such that, with S_0 recomputed from these inputs, $S_0 \xrightarrow{B} (s'\|\operatorname{cap} S_0)$ for some s', C opens to (S_0, B) and $N = G(\text{“null”}\|B)$; ctx $= (\text{mint}, C, pk_O)$. Words of any other length, such as $B\|(B')^i$ from Lemma 1, are not accepted.
5. L checks π_{mint}, the freshness of tag, that C is new and that N is unregistered, then stores the immutable mint record $C \mapsto (R, c, \text{tag}, pk_O)$, which fixes S_0 for all later proofs, sets the current owner of C to pk_O, and registers N.

Transfer $O \to O'$: O signs $(\text{transfer}, C, pk_{O'}, \text{nonce})$ with sk_O; L checks the signature and updates the current owner of C to $pk_{O'}$; the mint record is unchanged. Before that, O privately hands B to O' over an authenticated confidential channel, and before accepting (countersigning) the transfer O' recomputes S_0 from the immutable mint record of C and checks $S_0 \xrightarrow{B} (s'\|\operatorname{cap} S_0)$ for some s', $C = G(\text{“commit”}\|S_0\|B)$ and that $N = G(\text{“null”}\|B)$ is the registered nullifier of C. Ownership lives in L, not in knowledge of B: the old owner still knows B but cannot spend, because spending needs the current owner's signature.

Spend / redeem: the current owner O publishes π_{spend} (knowledge of the two-block B opening C with nullifier N, for the S_0 fixed by the mint record of C; ctx $= (\text{spend}, C, pk_O, \text{nonce})$) and a signature with sk_O on $(\text{spend}, C, N, \text{nonce})$, where pk_O is the current owner key of C; L checks the signature, π_{spend} against the registered N and that C is unspent, then marks C spent. Since N is public from the mint on, spending does not reveal it for the first time.

Bounty claim (for a protocol that uses reduced-round Keccak with capacity c): anyone presenting a proof of a *single-state* bridge at (R, c) for a tag fixed by the bounty contract collects the reward, and the contract raises the protocol's capacity to $c + 2$ (as in Protocol 1).

proven in zero knowledge, so c acts as a *denomination*. Bridges are not unique, since many exist for the same parameters, so the token is the commitment and not the bridge. Protocol 2 gives the life cycle. The circuit, the proof systems and an inheritance variant follow the protocol.

The bounty claim gives a self-enforcing, on-chain price for reduced-round security: while the reward is unclaimed, nobody has shown a bridge at that cost. Revealing a committed bridge later proves that the compute was spent before the commitment date (timed release, sealed bids).

Circuit and proof systems. *Avoiding f^{-1} in the circuit.* For a single-state bridge S the relation is: there are A, T with $f(S) = A$, $f(T) = S$ and $\operatorname{cap} A = \operatorname{cap} T$. The prover supplies $T = f^{-1}(S)$ as an extra witness and the circuit checks $f(T) = S$, so χ^{-1} (degree 3) never appears. The circuit is exactly 2 forward permutations, $2 \cdot 24 \cdot 1600 = 76{,}800$ ANDs, and 115,200 ANDs with a Keccak commitment to S; all constraints stay of degree 2.

Public statement. The token ID is a commitment and a nullifier

$$C = \text{TurboSHAKE128}(\text{“bridge-v1”}\|R\|c\|\text{tag}\|S), \qquad N = \text{TurboSHAKE128}(\text{“null”}\|S),$$

the latter published and registered at mint and checked again at spend time. In C, "bridge-v1" is encoded as 9 ASCII bytes, R and c as one byte each, the tag as 32 bytes and S as 200 bytes (243 bytes in total), and TurboSHAKE128 is used as in RFC 9861 [74] with domain byte `0x1F` and 32-byte output. Every proof binds a context ctx = (chain id, C, action, new owner pk, nonce) through the Fiat–Shamir hash (for Groth16 as a public input), so a copied proof is useless for another owner or action. *Limitation of the measured circuits.* The circuits of Tables 6 and 7 prove this single-state relation with C and N computed from S; they do not constrain S to depend on the tag, and they do not implement the MITM relation from a tag-derived S_0 of Protocol 2. With these circuits a witness (S, T) found once could be re-committed under any later tag with a new proof but no new search, so they do not by themselves give freshness; a ledger that registers $N(S)$ rejects such reuse, but it does not show that the search took place after the tag was published. Freshness requires deriving the start capacity from the tag inside the proven relation, which we have not implemented. A minimal alternative publishes only $\Delta = (A \oplus T)_{\text{rate}}$; it is binding only heuristically (≈ 1 solution per Δ, $\approx 2^c$ to forge) and is not hiding. Since minting a single-state bridge also costs $\approx 2^c$ (Proposition 3), this variant has no minting/forging gap; protocols that need the $2^{c/2}$ versus 2^c asymmetry should use the MITM-bridge keys of Section 4.2.1.

Table 6: Pairing-based proof systems for the bridge relations, measured on two machines. R1 is the one-permutation ownership relation of Protocol 4; R2 is a two-permutation single-state bridge relation (not the MITM relation of Protocol 2; the tag is not constrained inside it); R3 adds the TurboSHAKE128 commitment C (two calls of Keccak-p[1600, 12], via a locally patched gnark round range); R3+N adds a nullifier N computed from S (two more 12-round calls). All rows include the MiMC context binding (ctx). gnark v0.16.4 on BN254, median of 3 runs. Box: Intel Xeon (x86-64, AVX-512), 8 cores at 2.4 GHz; M5 Max: Apple M5 Max, 18 cores, 128 GiB. Box R1–R3 rows are from the box reference run, R3+N rows from a second run on the same box. Groth16 uses a single-party local setup and PLONK uses gnark's test SRS, which is fine for measuring cost but not for deployment. Groth16 proofs are 164 B in gnark's serialization (128 B plain). Code and logs: `06_zk_pairing_gnark/`.

			Box		M5 Max		
System	Constraints	Proof	Prove	Verify	Prove	Verify	Setup
Groth16 [35], R1	96,935	164 B	0.21 s	0.71 ms	0.070 s	0.71 ms	trusted
Groth16, R2	188,455	164 B	0.41 s	0.93 ms	0.126 s	0.75 ms	trusted
Groth16, R3	284,655	164 B	0.67 s	0.90 ms	0.240 s	0.79 ms	trusted
Groth16, R3+N	377,795	164 B	0.66 s	1.02 ms	0.248 s	0.70 ms	trusted
PLONK-KZG [30], R1	162,276	520 B	2.20 s	1.65 ms	0.84 s	1.14 ms	universal
PLONK-KZG, R2	317,218	520 B	4.19 s	1.59 ms	1.68 s	1.56 ms	universal
PLONK-KZG, R3	476,710	520 B	4.36 s	1.76 ms	1.70 s	1.17 ms	universal
PLONK-KZG, R3+N	631,334	520 B	8.06 s	1.95 ms	3.35 s	1.11 ms	universal

The measured constraint counts are 2.45× (Groth16) and 4.12× (PLONK) the AND-gate count of the circuit, because XOR and bit range checks are not free over a prime field; cost grows linearly in the number of permutations. The variant with the Keccak commitment C (three permutation-equivalents, R3) is also measured: it costs 2.46× (Groth16) and 4.12× its 115,200 ANDs, and the commitment adds about 95k Groth16 constraints, the cost of one full permutation.

What the measurements mean. With Groth16 the measured relations prove in under a second (PLONK: about 0.8–8 s, Table 6), and they verify natively in about 1–2 ms. Groth16 over BN254 can be verified on Ethereum through the pairing precompiles [68, 21, 22]; the four-pairing check alone costs $34{,}000 \cdot 4 + 45{,}000 = 181{,}000$ gas under EIP-1108, several times a plain

Table 7: Post-quantum, hash-based proof systems for the same relations (no ctx). Measured rows: a plonky3 [64] STARK with FRI over Goldilocks (Goldilocks^5 challenges, Keccak Merkle tree), zero-knowledge mode (hiding FRI), 8 threads, median of 5 runs, same two machines as Table 6. "proven100" is blowup 2^4, 40 queries, 24-bit grinding; "small" is blowup 2^5, 23 queries, 16-bit grinding. Security bits (conjectured/proven) are from plonky3's security model. Rows marked (est.) are sourced estimates, not measurements; no setup is needed for any row. †Per SHAKE256 evaluation: 55.5/80.1 KB at degree 2 and 14.9/21.1 KB with the degree-16 constraint trick (short/fast variants of the FAEST framework [6]); in [5] this SHAKE256 proof is a building block of a blind-signature scheme. Our relations need 1–3 evaluations; linear scaling to 1–3 calls is our extrapolation, and the security level is not stated in their table. Code and logs: `07_zk_pq_plonky3/`.

			Box		M5 Max	
System	Proof	Bits	Prove	Verify	Prove	Verify
STARK proven100, R1	1.17 MB	128/103	0.39 s	10.7 ms	0.12 s	4.3 ms
STARK proven100, R2	1.17 MB	128/103	0.50 s	10.7 ms	0.11 s	4.3 ms
STARK proven100, R3	1.89 MB	128/103	0.40 s	18.5 ms	0.17 s	7.3 ms
STARK small, R1	0.79 MB	128/73	0.13 s	8.6 ms	0.06 s	3.6 ms
VOLEitH for SHAKE256 [5] (est.)	15–80 KB†	–	not measured			
Binius64 [44] (est.)	100s of KiB	–	$\ll$ 1 s / 20–150 ms		not measured	
ZKBoo demo on f[200] [32] (est.)	tens of KB	–	not measured			
Commit-reveal (no ZK)	32 B + 200 B	–	μs to prove; 2 permutations to verify			

21,000-gas transfer. We have not deployed or measured an on-chain verifier. The asymmetry between creating and checking is large. Minting a key at $c = 80$ costs about 2^{41} permutation calls and forging one costs about 2^{80}, yet proving ownership takes less than a second and verifying the proof less than a millisecond: a bridge is costly to create, almost free to prove and check, and infeasible to fake. Verification at the millisecond scale also means that light clients, wallets, browsers and embedded devices can check ownership themselves, without trusting a server. Finally, the bottleneck is the hash search itself and not the cryptography around it, which supports the view that the scarcity of these objects rests on Keccak alone. The pairing-based backends are neither post-quantum nor Keccak-only, and Groth16 needs a trusted setup. A post-quantum, hash-based backend is also prototyped: the STARK of Table 7 relies only on hashing, needs no setup, and gives proofs of about 0.8–1.9 MB. Its 128/103 bits are plonky3's conjectured/proven *soundness* estimates for the proof system only; the end-to-end security of a token or signature is bounded by the witness search ($\approx 2^c$ classically, $\approx 2^{c/2}$ Grover queries, i.e. 2^{32} and 2^{16} for the $c = 32$ demonstration) and is not established. These proofs are generated in about 0.1–0.5 s with 8 threads and verify in about 4–19 ms. The protocols are therefore agnostic to whether the backend is pre- or post-quantum: the choice trades proof size (164 B versus about 1 MB) for assumptions (pairings and a trusted setup versus hashing only). A nullifier computed from the single-state witness (R3+N) is measured as well; it costs about one more permutation-equivalent (Groth16 proving 0.25 s on the M5 Max). On the M5 Max, proving is about 2.7–3× faster than on the box for both families, while pairing verification stays at about 1 ms.

Inheritance. Protocol 3 passes a token to an heir through the ledger. An heir can also receive a *child* bridge within Hamming distance d of the parent (or mined from nearby seeds), proven in ZK with the nullifier; generically this gives no shortcut over a fresh search, and a differential shortcut for low R is an untested conjecture.

Protocol 3 BridgeInherit: passing a bridge token to an heir (design only, not implemented; it relies on the tag binding of Protocol 2, which the measured circuits do not enforce, see Section 4.2).

Parties. Owner O (pk_O), heir H (pk_H), ledger L, optional escrow quorum Q.

1. *Mint:* O mints a token as in Protocol 2: a two-block MITM bridge B from the tag-derived S_0, published $(\text{tag}, C, N, \pi_{\text{mint}})$; L stores the mint record and sets the current owner of C to pk_O.
2. *Prove:* at any time, a fresh proof of the spend relation of Protocol 2 for C with $\text{ctx} = (\text{own}, C, pk_O, \text{height})$.
3. *Designate heir:* O posts a signed policy "$C \to pk_H$ after condition K"; L records pk_H as the heir key of C. The opening B must be delivered to H for redemption or re-keying: sealed for H by a time-lock puzzle [69] with verifiable encryption, or escrowed as threshold shares held by Q and released by a dead-man switch.
4. *Claim:* H publishes π_{claim}, a proof of the spend relation of Protocol 2 for C (knowledge of the two-block B opening C with the registered N, for the S_0 fixed by the mint record) with $\text{ctx} = (\text{claim}, C, pk_H)$, and a signature with sk_H on (claim, C, N); L checks K, that pk_H is the recorded heir key, the signature and the proof, and sets the current owner of C to pk_H.
5. *Re-key:* H spends C as in Protocol 2 and mints a fresh token by a new Protocol 2 mint with a new tag and pk_H.

Knowledge cannot be unlearned. Whoever ever saw a bridge can prove knowledge of it forever, so knowledge alone never gives exclusive ownership. This is why Protocol 2 binds the start state to pk_O and keeps ownership in the ledger, with an owner signature on every transfer and spend. Trusted hardware, or treating a bridge as a licence, are alternatives. The single-state bridge is a *mint certificate*; ownership lives in the ledger state plus key binding (every action needs the current owner's signature and a proof whose context contains that key), or in a nullifier plus a fresh single-state bridge.

4.2.1 A cheaper variant: publishing only the forward rate half

The keys of this section are MITM bridges (Theorem 1), not single-state bridges, in the notation of that theorem: $x \in \{0,1\}^c$ and $a, w' \in \{0,1\}^r$ with

$$\text{cap}\, f(a\|x) \;=\; \text{cap}\, f^{-1}(w'\|x) \;=\; v, \qquad f(a\|x) = (u\|v), \quad f^{-1}(w'\|x) = (w\|v).$$

Keys. The secret is w' (equivalently the full loop $L(y) = (y \oplus a, u \oplus w)$). The public data are x and the forward half a; hence anyone can compute $(u\|v) = f(a\|x)$ and in particular v. Publishing w or $(w\|v)$ would reveal w', since $f(w\|v) = (w'\|x)$ and f is a public permutation.

Minting. The owner runs the MITM birthday search of Theorem 1 at capacity x: about $2^{c/2}$ values $\text{cap}\, f(a\|x)$ against about $2^{c/2}$ values $\text{cap}\, f^{-1}(w'\|x)$, so about $2 \cdot 2^{c/2}$ calls (Remark 4). The owner knows w'.

Forging. From the public (x, a), a forger must find *some* w' with

$$\text{cap}\, f^{-1}(w'\|x) = v,$$

a c-bit target preimage under f^{-1} restricted to the 2^r inputs $(w'\|x)$. For an ideal permutation each trial succeeds with probability $\approx 2^{-c}$, so forging costs $\approx 2^c$ calls. The birthday trick does not apply, because v is fixed by the public data. Solutions exist in abundance: since $2^r \gg 2^c$, about 2^{r-c} values w' satisfy the condition, and the owner knows one of them. The key is

Protocol 4 SplitBridge: keys from MITM bridges with only the forward rate half public (design; the ownership-proof circuit is prototyped and measured, Tables 6 and 7).

Public parameters. Permutation f (R rounds), capacity c, rate $r = b - c$; optionally a threshold $t \le n$.

KeyGen (owner O):

1. Run the MITM birthday search of Theorem 1 at a capacity value x (for example x derived from a fresh public tag), obtaining $a, w' \in \{0,1\}^r$ with $\operatorname{cap} f(a\|x) = \operatorname{cap} f^{-1}(w'\|x)$ ($\approx 2 \cdot 2^{c/2}$ calls).
2. Output $\mathsf{pk} = (x, a)$ and $\mathsf{sk} = w'$. Never publish w or $(w\|v)$, since $f(w\|v) = (w'\|x)$ reveals w'.

Verify a secret $\tilde{w}$ against pk: accept iff $\operatorname{cap} f(a\|x) = \operatorname{cap} f^{-1}(\tilde{w}\|x)$ (2 permutation calls).

Prove ownership without revealing: give a ZK proof of knowledge of (w', w) with $f(w\|v) = (w'\|x)$, where the verifier computes $v = \operatorname{cap} f(a\|x)$ (one forward permutation in the circuit, 38,400 ANDs at 24 rounds), with a Fiat–Shamir context binding the action. Without ZK: reveal w' (commit–reveal); a revealed w' cannot be unlearned, so the key is then spent.

Split (t-of-n): share w' with Shamir (or XOR for $t = n$) among n custodians. **Reconstruct:** any t custodians combine their shares into $\tilde{w}$, and anyone runs **Verify**, so the reconstructed secret is publicly checkable against (x, a) with no extra commitment.

Forging cost for a given pk: a c-bit fixed-target preimage under f^{-1}, $\approx 2^c$ calls (heuristic, ideal-permutation model), versus $\approx 2 \cdot 2^{c/2}$ for KeyGen; hence $c = 32$ is demo-only.

therefore not unique, and none of the constructions below needs uniqueness. *For comparison*, the single-state variant with public P_1 and secret x (relation $\operatorname{cap} f^{-2}(P_1\|x) = x$, Definition 3) has ≈ 1 solution per P_1, but minting it costs $\approx 2^c$, the same as forging, so it has no minting/forging gap. Experiment for that variant ($f[200]$, $R = 18$, $c = 16$, exhaustive over all 2^{16} values of x for 20 random P_1): the counts were 0,1,4,0,1,1,2,1,0,1,0,2,0,2,1,0,1,2,1,0, with mean 1.00, and 7 of 20 values of P_1 had no solution ($e^{-1} \cdot 20 \approx 7.4$). This is empirical and small-sample.

Ownership proof. The statement is "I know w' with $\operatorname{cap} f^{-1}(w'\|x) = v$", with public (x, v), or (x, a) from which the verifier computes $v = \operatorname{cap} f(a\|x)$. The witnesses are w' and w, and the circuit checks $f(w\|v) = (w'\|x)$. That is one forward permutation (38,400 ANDs at 24 rounds) with no f^{-1}, and no commitment is needed, because (x, a) is itself the binding public tag. If instead the circuit also recomputes v from (x, a), it is two forward permutations (76,800 ANDs). A commitment to bind auxiliary data costs a further 38,400 ANDs. The simpler option uses no ZK: commit-reveal of w', which anyone checks with 2 permutation calls ($f(a\|x)$ and $f^{-1}(w'\|x)$).

Split knowledge. Given (x, a), any revealed w' is checked directly. A genuine t-of-n scheme shares w' (XOR or Shamir shares). The bridge supplies *public verifiability*: any reconstructed w' is checked against (x, a) with 2 permutation calls. The result is a self-verifying secret whose public tag (x, a) is binding with no extra commitment, in the sense that producing *any* valid secret for it costs $\approx 2^c$. Protocol 4 collects the steps above.

A deployed instance. The one-sided challenges c88-B3 and c88-B4 of Section 2.7.1 are SplitBridge public keys (x, a) at $c = 88$ on the full Keccak-$f[1600]$, with the secret halves kept by the author and committed with a salt; their forging cost is about 2^{88} calls (heuristic).

Uses beyond inheritance. Escrow, key recovery, sealed bids, timed release and other uses are listed in Section 4.2.3. *PoW receipts and bounties*: minting costs $\approx 2 \cdot 2^{c/2}$, while forging for a given ID costs $\approx 2^c$, a quadratic gap. *Dead-man switches*: release shares of w'. *Whistleblower and revocation tokens*: revealing w' proves authorship or revokes. *Splitting among* 2^k *heirs* (remark): the 2^k messages of a bridge-amplified multicollision can serve as distinct public labels

that all hash to the same digest. They are not secrets, because anyone who knows (A, A') and B can derive all of them. Each heir must therefore also hold a private share of w' bound to their label. Reduced-round variants may be weaker. Hiding of w' given (x, a) is not proven. Since many w' are valid, a sealed-bid or commitment use that needs the *specific* w' to be binding must add a commitment to w'.

4.2.2 A zero-knowledge signature from split bridges

Relation and keys. With the notation of Section 4.2.1, let

$$R = \{((x, v); w') : \operatorname{cap} f^{-1}(w' \| x) = v\},$$

where $v = \operatorname{cap} f(a \| x)$ is computed from the public (x, a). Key generation runs the MITM birthday search of Theorem 1 at capacity x ($\approx 2 \cdot 2^{c/2}$ calls) and outputs $\mathsf{pk} = (x, a)$ ($c + r$ bits) and $\mathsf{sk} = w'$ (r bits). The prover proves knowledge of the witness (w', w) satisfying $f(w \| v) = (w' \| x)$. This is one forward permutation, i.e. 38,400 AND gates for Keccak-f[1600] at 24 rounds (76,800 if the circuit also recomputes v from a), and no f^{-1}. Any MPC-in-the-head proof system (ZKBoo/ZKB++ [32, 23], KKW [48]) or VOLE-in-the-head [7] (FAEST-style [6]) applies, and Fiat–Shamir yields a signature.

Security. Completeness, special soundness (ZKB++: error 2/3 per repetition, so $\lceil \lambda / \log_2 1.5 \rceil$ repetitions) and HVZK are standard. By the forking lemma, a forger yields some w'' with $((x, v); w'') \in R$, i.e. it solves the fixed-target problem $\operatorname{cap} f^{-1}(w'' \| x) = v$. In the ideal-permutation model that costs $\approx 2^c$ classically and $\approx 2^{c/2}$ Grover queries (heuristic). This is a classical random-oracle argument; a proof in the quantum random-oracle model, a concrete EUF-CMA theorem and its reduction loss are left open, so the scheme is a candidate. Uniqueness is not needed: about 2^{r-c} witnesses exist for each key, and the forger may output any of them.

The asymmetry. For an honestly generated fixed public key, the estimated classical generic witness-search cost is about 2^c, while key generation costs $2^{c/2}$. If one takes this witness-search cost as the target security level, one may set $c = \lambda$ and then key generation costs $2^{\lambda/2}$. In concrete terms: $c = 80$ costs 2^{40} (demonstration level); $c = 128$ costs 2^{64} (feasible only as a large one-off computation) and gives a Grover *query* bound of $\approx 2^{64}$ (see the remark below on realistic quantum cost); $c = 256$ is infeasible. A plain one-way function such as $w' \mapsto \operatorname{cap} f^{-1}(w' \| 0^c)$ gives the same 2^c bound with trivial key generation. The bridge adds proof-of-work-minted identities and the link to the tokens of Section 4.2.1, not extra security.

Sizes (estimates). For $c = \lambda = 128$, published size formulas give ≈ 2.2 MB (ZKB++, 219 repetitions), ≈ 0.7 MB (KKW, $N = 16$, $\tau = 36$) and ≈ 106 KB (VOLEitH [7], $\tau = 11$). For comparison, FAEST-128s needs ≈ 5 KB and SPHINCS$^+$-128s 7.9 KB. These figures are for the two-permutation circuit of the earlier single-state design (76,800 ANDs); the one-permutation relation above needs about half the circuit, so they are upper bounds. The scheme is not a speed or size champion. It is a proof of concept that the bridge hardness yields a Keccak-only signature.

Remark (asymmetric minting). Key generation costs $\approx 2^{c/2}$ calls of f, while forging a key for a given public (x, a) costs $\approx 2^c$ classically (heuristic). An entity with very large parallel compute (a large company or a state, say 2^{64}–2^{80} calls) can therefore mint identities with $c = 128$–160. Everyone else cannot forge them, including peers with the same budget. Minting thus becomes a costly, publicly verifiable act, a "proof of capability" that relies only on Keccak. Two limits apply. The gap is only quadratic, so an adversary with the square of

the minter's budget can forge. Against quantum adversaries, Grover's algorithm needs $\approx 2^{c/2}$ *queries*. Realistic cost estimates are much higher, however, because of depth limits (NIST's MAXDEPTH bound of 2^{40}–2^{96} [63]), the poor $\sqrt{P}$ parallel speed-up [75], and the circuit size of the Keccak oracle; for SHA3-256 preimages, Amy et al. estimate $2^{166.5}$ logical-qubit-cycles rather than 2^{128} queries [1], and depth-limited AES-128 key search is analysed in [45]. Whether the asymmetry persists against depth-limited quantum adversaries for this relation is open.

4.2.3 Other potential applications of Keccak bridges

Protocols 1–4 are initial proposals. They are not fully verified, and apart from the zero-knowledge circuits of Protocols 2 and 4 (Tables 6 and 7) they are not implemented, since a complete security analysis and implementation of each would make the scope of this paper too big. The further uses listed below are only sketches, and we deliberately leave them open. We invite the community to turn each of them into a fully specified protocol with a security model and proofs. Concretely, the open problems are:

1. **Hiding.** A formal hiding notion for w' given (x, a), and a reduction in the ideal-permutation model.

2. **Sealed bids and timed release:** protocols that use (x, a) as the binding public tag and reveal w' later (e.g. combined with time-lock puzzles [69]).

3. **Threshold custody, escrow and key recovery** with shares of w' and publicly verifiable reconstruction of w'.

4. **Bounty and proof-of-work markets** that exploit the $2^{c/2}$ versus 2^c gap between minting and forging.

5. **Revocation and whistleblower tokens.**

6. **Smaller post-quantum proofs** for the one- or two-permutation ownership statement. Our plonky3 STARK (Table 7) already gives a post-quantum, Keccak-only prover, but its proofs are about 1–2 MB; the open direction is much smaller post-quantum proofs, for example with VOLE-in-the-head systems such as FAEST [5].

7. **Ring and group signatures.** Keccak-only ring and group signatures: each member i holds a private MITM-bridge secret w'_i for a public (x_i, a_i), a signature is an MPC-in-the-head OR-proof of knowing one witness, and a group manager can open signatures if the signer additionally encrypts their index.

8. **Anonymity from a shared secret.** If all members share one bridge-amplified multicollision (or one bridge), any valid proof shows only that *some* member signed. This perfect but untraceable anonymity suits whistleblowing within a defined group. For anonymous voting, the scheme must add per-member nullifiers to prevent double voting, which a shared secret cannot provide.

9. **Anonymous bidding with certified multicollision tickets.** The organizer signs distinct labels from a bridge-amplified multicollision, and each signed label serves as an entrance ticket and one-bid nullifier. Knowing (A, A', B) alone does not let anyone mint a ticket. Unlinkability between bidder and bid requires blind or anonymous ticket issuance, since otherwise the organizer can link each label to its holder; a Keccak-only blind issuance remains open.

10. **Dual-capacity commitments and message-bound proof of work.** Publish for a message M both $h = \text{SHA3-256}(M)$ and a reduced-capacity digest $h_c =$

Keccak[r=1600$-c, c$](M). A loop L found with $\approx 2^{c/2}$ calls (Theorem 1) gives every $M'_i = C_0 \| L(y_0) \| L(w')^i \| C_1 \| (M_0 \oplus y_1) \| M_1 \cdots M_k$ ($i \geq 0$, $|M| \geq r$) the same reduced-capacity digest h_c. This does not give them the same ordinary SHA3-256 digest: h still binds only the original document M, and the M'_i have pairwise distinct SHA3-256 digests (as is already the case for the 27 messages of Section 3.4.1). Opening the original document (a preimage of h) and demonstrating knowledge of a reduced-capacity bridge (a second preimage of h_c) are therefore separate statements; equality under ordinary SHA3-256 does not follow. Moreover the same loop can be reused across documents, so an argument that the work is document-specific and fresh is still needed. The constructions suggest proofs of work bound to a document, and bridge-holder signatures anchored by h (Section 4.2.2), but the security of each protocol, in particular binding of h jointly with h_c, remains to be formalized.

11. **Hidden counts under SHA3-256.** In the setup of the previous item, the digests $H_j = \text{SHA3-256}(L(w')^j)$ hide L by preimage resistance (at $c = 512$ the word L is not a loop), but they are not linked to each other, so a statement such as "$j \leq i$ and L loops at capacity c" needs a ZK proof of about two permutation calls per block. A chained variant $H_{j+1} = \text{SHA3-256}(H_j \| L)$, or committing to the internal state after L^j, gives commit-then-reveal-steps verification once L is opened.

12. **One-sided reveals inside the capacity-c sponge.** There all M'_i share h_c, so i is hidden by construction, and publishing only the forward half a of L (the analogue of P_1) fixes $v = \operatorname{cap} f(a \| x)$, while completing the loop requires w' with $\operatorname{cap} f^{-1}(w' \| x) = v$, a single-point condition costing $\approx 2^c$. The owner opens with w' (two permutation calls) or proves knowledge of w' in zero knowledge, so committing costs $\approx 2^{c/2}$ (Theorem 1), forging an opening from a costs $\approx 2^c$, and binding to M comes from h.

13. **Bridge-rooted zero-knowledge signatures.** A secret bridge yields 2^t distinct second preimages $M'_0, \ldots, M'_{2^t-1}$ of a fixed message M. The signer builds a Merkle tree whose leaves are hiding commitments $\ell_i = H(i \,\|\, M'_i \,\|\, \rho_i)$ with fresh randomness ρ_i, and publishes only the root as the public key. To sign m, the signer takes an unused index i and outputs (i, π), where π is a zero-knowledge proof (Section 4.2) of knowledge of (M'_i, ρ_i) and an authentication path such that ℓ_i lies under the root, $M'_i \neq M$ has the same digest as M, and the proof is bound to $H(m \,\|\, i)$ (for example as a signature of knowledge). Leaves are never opened: since a single M'_i exposes the connectors and the loop, and hence all others (Section 2), revealing one leaf would give away the whole key. The root fixes a public, countable budget of 2^t signatures; verifiers reject a repeated index, and the signer can revoke the key by publishing the root as spent. Forging requires either a second preimage of M, or a collision with M together with a valid leaf opening under the published root, or breaking the proof system or H. A bridge at the same capacity value x does not by itself supply the committed leaf opening, its randomness ρ_i and an authentication path under that particular Merkle root; whether, and at what cost, such an opening can be produced from a bridge is left open. Leaves that may be opened in the clear need independent secrets: with one bridge per leaf, for instance at capacity values $x_j = \text{PRF}(seed, j)$, opening leaf j exposes only the bridge at x_j, but each leaf then needs a separate search. Theorem 4's growing-table cost fixes one endpoint capacity x and gives no estimate for collecting loops at K distinct endpoint capacities; a cost claim of the form $\approx \sqrt{K} \cdot 2^{c/2}$ would need a model that checks both endpoint closure and the committed-leaf relation, and is left open. A single bridge thus gives exponentially many leaves only behind zero-knowledge proofs, and independent bridges give polynomially many leaves that may be revealed. Unlike the split-bridge signature of Section 4.2.2, the key here is one bridge and the budget is explicit in the tree. Precise definitions, the choice of c against the $2^{c/2}$ loop

route, and a security proof are left open.

To support this, the full $f[1600]$, $c = 32$ test vectors (Appendix D), the generators and checkers, and the SageMath sources of the code release are released with the paper, so that every new protocol can be prototyped and checked against concrete instances.

5 Conclusion

In 2011 we observed that one path in the Keccak sponge from a state whose capacity has all zero bits back to such a state gives free second preimages of every message of at least one full rate block [34]. This paper makes that observation explicit at an arbitrary capacity value. The two-block meet-in-the-middle loop of Theorem 1 gives the same unlimited family of second preimages from any capacity x, together with the repetition lemma, the extension to k-block loops and to cycles of the capacity map (Theorem 2), and the single-state bridge of Proposition 3. The same objects give cheap collisions and bridge-amplified multicollisions (Theorem 3), which need one inner collision and one connector instead of the k collision searches of Joux's construction [46]. These statements are constructions with elementary proofs; the core identities (Propositions 1 and 2), the soundness of the abstract literal search and that of the checker for birthday hits as two-point connectors are also machine-checked in Lean 4 for an abstract permutation (Appendix A). What is not proven are the costs: about $2 \cdot 2^{c/2}$ for a loop and 2^c for a single-state bridge are heuristic in the random-function model, and they agree with the generic $2^{c/2}$ inner-collision bound of the sponge [12, 52], so the loop is optimal up to constants for an ideal permutation.

The experiments support these heuristic costs and, apart from the cells reported in Appendix E, we detected no departures from generic behaviour. On Keccak-$f[200]$ for $R = 1..18$, the literal search, the birthday connector search up to $c = 32$, the fixed-capacity MITM bridge search for $c \leq 24$, the distinguished-point search up to $c = 48$ and its parallel version on $f[200]$ and on the full Keccak-$f[1600]$ match the predicted constants. The exhaustive $c = 32$ campaign (36 functional graphs and the fixed-rate sweeps) shows, apart from the degenerate $R = 1$ and the single known $R = 2$ inverse map, no deviation from random-mapping statistics after correction for multiple testing; the small excess of short cycles in the sweeps disappears once $F(1000)$ is calibrated on random functions. The 286-cycle of that $R = 2$ map gives a repeatable message mode, but it is a lucky instance of a small capacity and not a weakness of Keccak. All constructions are instantiated and independently verified on the full 18-round Keccak-$f[200]$ and the full 24-round Keccak-$f[1600]$ with $c = 32$: second preimages, a 2^8-multicollision, preimages at $\approx 2^{17}$ calls, and the step-by-step worked example of Section 2.4. These costs are generic in c. SHA3-256 itself is not weakened: with $c = 512$ the same procedures cost $\approx 2^{256}$, and we claim no attack on full Keccak or SHA-3.

At $c = 88$, four bridges on one generalized capacity arrived in line with the Gamma law of Theorem 4; two of them are now one-sided forgery challenges with heuristic generic cost 2^{88} and 2^{92}, and the downward-closure lemma explains their free upgrades to a larger effective capacity. Beyond the horizon, the generic hardness becomes a resource. The gap between minting a bridge ($\approx 2^{c/2}$) and forging one for given public data ($\approx 2^c$) underlies the protocols of Section 4: proof of work with hardness-adaptive capacity, bridge tokens, SplitBridge keys, a zero-knowledge signature from split bridges, and a bounty market for reduced-round security. To our knowledge, these are the first unkeyed Keccak-based protocols that deliberately use a small capacity, so that the generic bridge cost is a feasible and tunable hardness parameter. The ownership circuits are prototyped and measured: pairing-based proofs of 164 B that verify in about 1 ms (Table 6), and post-quantum, hash-based STARK proofs of about 1–2 MB (Table 7). The asymmetry also scales with resources: an individual with modest hardware can mint a bridge at $c = 88$ ($\sqrt{\pi} \cdot 2^{44} = 2^{44.83}$ calls, the heuristic constant of Table 5) in about 1.5–3 days,

depending on per-thread throughput, on our 160-thread setup (an Apple Silicon M5 Max Pro laptop plus two servers), while the estimated classical generic cost of forging a given published bridge is about 2^{88}; under the throughput assumptions of Section 4, a sufficiently resourced issuer such as a state could mint bridges at $c = 160$ (about 2^{80} calls), while the estimated classical generic cost of forging a given one is about 2^{160}, which would give them a role similar to state-issued bonds. The gap is only quadratic, the protocols are design proposals, and their security in a formal model is not established.

Several questions remain open. They include a formal hiding notion for the SplitBridge secret, much smaller post-quantum proofs, and full specifications with security proofs for the sketched applications of Section 4.2.3, among them dual-capacity commitments and bridge-rooted zero-knowledge signatures. On the cryptanalytic side, the algebraic bridges for $R = 2$, whose cost grows below the generic rate between $w = 8$ and $w = 16$, need a baseline, a $w = 32$ point and the $R = 3$ case.

Acknowledgments

I initially discussed the ideas of this paper with Google's Gemini AI. However, very soon I hit the security-guardrail blocks caused by the cryptographic nature of the research. Then I moved the work entirely to Grok Bot. Grok Bot, working as a team of assistants, contributed to the discussion of attack designs; wrote the C, Python, SageMath, Lean, Go and Rust code; ran the experimental campaigns (Fig. 1 bridges, literal and birthday bridge searches, functional-graph and cycle searches, and the calibration of $F(1000)$); built the zero-knowledge prototypes (Groth16 and PLONK in gnark, a STARK in plonky3); checked the literature and the citations; and drafted the LaTeX text. OpenAI Astra carried out the technical review and proofreading of the final manuscript and, with reproducible counterexamples, suggested corrections that we adopted, notably distinguishing birthday-search connectors from true bridges and tightening the protocols.

A Artifact and formal proofs

Our code release, the folder `beyond_event_horizon_code/` (see its `README.md`, `HOWTO.md` and `run_all_tests.sh`), is archived on Zenodo [33] (concept DOI 10.5281/zenodo.23727976 for all versions; this paper describes version 1.7, version DOI 10.5281/zenodo.24163450; version 1.6.1 has the version DOI 10.5281/zenodo.23918542) and contains:

- `01_keccak_reference/`: Keccak-$f[200]$ and Keccak-$f[1600]$ references in C and Python with exact inverses, checked against XKCP test vectors, and bridge verifiers.
- `02_bridges_fig1/`: `general_bridge_examples/` (Remark 6, Section 2.4), `f1600_c32/` (generator, standalone checker, vendored XKCP `CompactFIPS202.py`, logs and test vectors for Section E.4 and Appendix D), and `tmto_dp/`: the distinguished-point (van Oorschot–Wiener) claw search `pcs.c` (with `tm.h`), its multithreaded version `mt/pcs_mt.c`, and the proof-of-concept bridges and checker logs in `dp_poc/`.
- `03_campaigns/bridge_cycle_search/`: `bridge_cycle` (literal and birthday searches, C with a Python twin) and `mitm_fixed_cap.py` (fixed-capacity MITM bridges), with the campaign scripts.
- `04_cycles_and_calibration/`: `fg_rounds`, the fixed-rate sweeps, the 286-cycle and the $F(1000)$ calibration; self-tests and known-answer tests pass.
- `05_formal_sage_lean/`: in `sage/` readable SageMath and Python sources (`bridge_explorer.py` with a settings block for $b \in \{200, 400, 800, 1600\}$, R, c; the reference `keccak_p.py`

with exact inverse; symbolic bridge equations; no caret operator, so the files are Sage-preparser safe); and `lean4/`: a Lean 4 (v4.35.0, core library only) formalisation, built with `lake build`, without `sorry`.

- `06_zk_pairing_gnark/` and `07_zk_pq_plonky3/`: the zero-knowledge prototypes of Tables 6 and 7.
- `08_rp_control/` (new in v1.2): the random-permutation control of Remark 7, the distinguished-point search `dprp.c` with interchangeable Keccak-$f[1600]$, true-random-permutation and SHAKE256-Feistel backends, the 1485 bridges with their independent checker, and the result tables.
- `bridges/c80/` and `challenge/` (new in v1.2): the $c = 80$ bridge with a standalone verifier, and the capacity-bridge challenge of Section 2.7 (verifier, submission format, rules, record table and tests).
- `09_distributed_search/` (new in v1.2, extended through v1.6.1): the distributed distinguished-point search that found the $c = 80$ bridge (workers on many machines, one read-only collector over ssh with a complete SQLite table, JSON presets up to $c = 88$, a local simulation), with start points from fresh $\max(2c, 256)$-bit seeds and a $\max(2c, 256)$-bit walk key, and the campaign story. Version 1.6 adds an explicit IV-transition mode ($0^c \to x$, with the transition kind recorded in the output), a balanced default $p = \frac{1}{2}$, `assemble_message.py` and `verify_message.py` for k^L messages from a standard-IV prefix, from the empty prefix at $x = 0^c$, or from a generalized-capacity start $0^r \| x$ (two independent Keccak implementations, a $c = 512$ check against `hashlib.sha3_256`, and a reconstruction of each word from its σ label and the named bridges), and a hardened collector (clean shutdown on STOP and signals, one supervisor per run, schema migrations, restrictive file modes, call accounting across worker restarts); these produced Section 3.4.1.
- `bridges/c88/` (v1.6.1 as `bridges/c88_kL8/`, extended in v1.7): the two public $c = 88$ bridges, their 8 messages, the run records of all four bridges and `gamma_c88.py`, which reproduces Table 4 and Figure 3.
- v1.7 additions to `challenge/`: the verifier accepts every integer capacity (bit-level split, as in the distributed search) and reports the effective capacity (`-ceff`, `ceff.py`; Lemma 5, Remark 12); `challenge/one_sided/` holds the one-sided challenges of Section 2.7.1, the public verifier `verify_one_sided.py`, `open_commitment.py`, the generic tool `make_one_sided.py`, and examples with public openings, including a toy $c = 32$ instance with three different completions.

Assumptions of the Lean model. The state is `BitVec r` × `BitVec c` (rate first, capacity second) and f is an *abstract* permutation (a structure with `toFun`, `invFun` and both inverse laws). No Keccak internals are modelled.

Proven in Lean. $f(f^{-1}(S)) = S$; Proposition 1 (`bridge_identity`) and its loop corollary (`bridge_loop`); Proposition 2 (`two_point`); soundness of a fuelled literal search (every returned state is a single-state bridge); soundness of a checked birthday hit (the word $(0^r, \Delta)$ maps S_i to S_j; this is a two-point connector, and a bridge only if also cap S_i = cap S_j, which the Lean lemma does not assume). Connectors are `IsConnector` and `IsTwoPointConnector`, single-state bridges are `IsSingleStateBridge`, and the corollaries `two_point_isTwoPointConnector`, `checkHit_isTwoPointConnector`, `bridge_iff_loop` (S is a single-state bridge iff $(0^r, \Delta)$ is a two-point connector from S to S) and `kblock_loop_isConnector` (a k-block loop is a connector from x to

x) are proven. The algebraic parts of Theorem 2 (`kblock_loop`, `kblock_repetition`, `kblock_loop_pow`, `kblock_insert`, `kblock_second_preimage`) and of Theorem 3 (`cheap_collision`, `multicoll_same_state`, `multicoll_length`, `multicoll_injective`: 2^k distinct equal-length block words reaching the same state) are proven at the level of block words, with no `sorry` and only the axioms `propext` and `Quot.sound`.

k^L multicollisions in Lean. `KLMulticollision.lean` models the sponge with an arbitrary function f on `BitVec r` $\times$ `BitVec c` (not the abstract permutation of `Bridge.lean`), which suffices once the bridge equations are given as hypotheses. It proves free rate absorption, composition, injectivity of the words, the length $|P| + 2L + 1$, the common full state $f(z\|x)$, equal outputs under any common suffix, and the count k^L of words (`kL_count`); the axioms used are `propext` and `Quot.sound`, and `kL_count` additionally uses `Classical.choice`. It does not prove the existence of bridges, any search cost, Keccak correctness or padding.

Incremental search in Lean. `IncrementalSearch.lean` models the collector of Algorithm 3 as a filter on an arbitrary stream of candidate blocks (P_1, M, P_3): a candidate is appended only if the two-call check succeeds and its first half P_1 is new. It proves that every reported candidate is a bridge on x in the sense of Definition 4 (`collect_isBridge`, Lemma 2(i)), independently of how the candidates were produced and after any number of resumptions (`collect_resume`); that the first halves P_1 are pairwise distinct (`collect_firstHalf_nodup`, Lemma 2(ii)); and that the collector output supplies both hypotheses of Theorem 4 (`collect_to_kL`). The axioms used are `propext` and `Quot.sound`. Completeness of the search and the cost $E[T_{k+1}]$ are not formalized.

Not proven. The capacity-map cycles of Theorem 2(ii), and the padding and message-level validity in Theorems 2(iii) and 3. All cost statements (2^c, $0.886 \cdot 2^{c/2}$, and $E[T_{k+1}]$ in Theorem 4) are heuristic in the random-function model; the hash-table/replay logic of the C and Python code, the correspondence between code and Lean model, completeness of the searches, and every experimental statement are outside the formalisation.

B Vectors of the $c = 80$ standard-IV 3^3-multicollision

Full 24-round Keccak-$f[1600]$, $r = 1520$, $c = 80$, capacity $x =$ `73e10d5cd30ae96a5434` (Section 3.4.1). Each object is three rate blocks of 190 bytes (hex, FIPS 202 byte order) and the intermediate capacity y. For the transition T, absorbing P_1 and M from the all-zero IV gives the state $(P_3\|x)$. For a bridge B_i, in the notation of Definition 4, $a_i = P_1$, $v_i \oplus b_i = M$ and $w_i = P_3$: from $(0^r\|x)$ the blocks P_1, M lead to $(P_3\|x)$. The message for $\sigma \in [3]^3$ is P_1^T, M^T, $P_3^T \oplus P_1^{\sigma_1}$, M^{σ_1}, $P_3^{\sigma_1} \oplus P_1^{\sigma_2}$, M^{σ_2}, $P_3^{\sigma_2} \oplus P_1^{\sigma_3}$, M^{σ_3}, $P_3^{\sigma_3}$ (the last block is t_σ with $z = 0^r$), followed by the SHA3 padding block `06 00...00 80`. All 27 messages are in the file `kL27_c80.jsonl` (sha256 `277d0ece...22bb`), and all give the SHA3-256-structured ($c = 80$) digest `14a4d530cdfdf137a60719c662f1a64f889d832583d512f0311adfa8040f6fc7`.

Transition T $(0^{80} \to x)$. $y =$ `c1859fd5ffea04084956`.

```
P1 = c60db63cbbb2076a6663fff8f523fc33817ffe165775c17d25e7bb4d4d2eb54dce59fd041b03
     c5a691af3c13f0973b0879d23123d05fdf5ae43d1e624e79535ce150f5579b8f19a64cbee976
     cb91de56ac795fa1284eb02ba33bb3fde05b8c6502884c9df95f930906202264430a6b414f19
     29756cd1dc8f7d6d04fd0f6df8e5f63976f9e99a78288ba52bcaf87046e19760d2422c01cd7c
     2a3b54841cc7d4b23fcc850ca3bdce8f468f802110e48b59664e4b43247a1e5405060664b762
M  = d12dda1f7f498864bb46357d8d808218d2799125e8b367f0e4900d5e84c0e0c1dd9643772759
     b734cf6edf39e28062abdc8cd959c31c7d50733af7e3f9976bb78869b7469609adb07f4ce205
     2b82d34b983cb8a756e795b7c578bfcf53bb20a45b7e54dbd8ac89534bf2d5b30385e219edb2
```

```
     4bb1aea12e18563c6ccb1d13a1c2c162cfa2b414c9b08b20ef2f1c47421965ab570850734cb3
     294d32107f48d64d7b36b598e98cd4333409f7a3e3263d1e3acf9d2d0b5431191dab5c9e8848
P3 = 66901d111e48e38b7e1586fcdc67859f19da6cfb241fdf6efcbb9074fa378c1d4f4e1e7b180b
     90e7288b600895e6d7d4c723ae70f4a705463fd2a9da16bc4d8525c8a96c896436d1ac14fdbb
     bcf2de6b412eee2f813136ad79a79488275938484e817aa5b8f18ad727981c8bdf4f028b80e9
     198e0a6c7c37cc3019d73dc509c9ae099841d0a59e7b092600b72ec242a6a4489adeac67b463
     5362a7c771ce0c6cac50edb9a67ec895cd46711fdf9cb890132c1c2391d2c43c7c88a48baba1
```

Bridge B_1 (found second). $y =$ ee95a4152637611daf26.

```
P1 = 050377cecb5763894837587384869b0b8cf0dfe963b93f29486272cde710ec8ce2e78798c9a1
     b2b015e73cabefb0e1ce29302580419d6612baa7d57ace0391510915521cb7936a15111a335e
     89d012a3748dbd59930ea6816e5ecd0371e4fcff34c57c43a7c6ab3d044f95303aeeb8380bf7
     c7c84d69bd99ea6e8e96b25ceee0cf6b3449685b044649a66eece3ead6abf9a07b1dd563556b
     63ce37cb0e8521e51eeedde673233d52cbf53f4e67d65a2bf3deba8ad6ab408fa6642b699128
M  = f6eb9066185e22553c465fdea6d2f93ec844b56e83ef6a1842be7316591a759c2fa4f722a71d
     2bc221421b23399237cdd12dba0752666324c45edfcf86a2ebe3dc26e52397d76f5fbe19d2b7
     d7a49165cc2c0fac340e9c096315aa30b0b44dd3642fbbfd31d79e7207f18d65eef81d8c21ca
     d126e5b99e3ccfb98e29f15fdbb6fc6ca7acc2fb4a35eb1862fca3252ba3cbe1b57615e1b8f2
     621a391e08b365005142d4684ac2f8deaaffb7ca2f29783b1d10d44f4ae71cd6634c27a16b6f
P3 = 7d60189265f3276f4529253e352487eb5fbc2804f7d0f57e7a3996b63b3b8d786385459f730f
     80412c68678bc54d562db064def917e3a0d2e5c8264954b2dfd7b2d1196215a6834ade9173b0
     4cef348c43611ea56e5d5317c93a2f5519797f7bfec4f62195be4b2fcc3e07571ef5180b4136
     a09ad8aea9687c3cc8d540bea00cf0a3856e70b16aa0982b62e4c9d8e93904195e66252a72bf
     47f22f7fd9edadedddd430901ad69a12195b9eaf5490466a87b50bbde84c87b3dd74e564c01d
```

Bridge B_2 (found third). $y =$ 851b3543804e3f48702a.

```
P1 = d46bf6b3b2935489c3021d682fb937fdcedbac6d40d67a48604c0b668e59d3a05cbf764c6937
     e94e7bf785f776c6caace83270ec2b251dad46bbfd3dbd18fdede1730e39bdeff66b3ce484a0
     8bc3fe72acc1b028a925adcf52eb59b46eba8182ca8fde5f94886053010252a44df50ff26782
     d77942c6d69c62581e59bf969c53d8431ab9af96fd57488f6962f175e0032a22574739ea7752
     8a5235616ef89802d2b9cab69233fc4d291932fa095096bd2399233dccbc21813e05c64c4ca5
M  = 90cb05be83e330af156536c9774a5042d9fb942bd81f6d0559f88c2070fae2f609c67d4465ac
     5258dc4e845574c685695dc14781f693ca523e70dbc37378f7e3c132a19bec77a541f2e534fd
     4aa0b6b7d54f13fc5e0e9b1048e5e6f24b03e8bad89a30459b5086d1a61dca8d409ecbe32d25
     c98cce6a627c7216a022111df143bba6f5095825bbf516ad9cea0b741d246ff6db4057924656
     af52fa2be30d33c369d5aa32deb33896e67858888f56d0cd8412446eedd379c223cc03b9d871
P3 = a9da115988101721905217fae3e13ae50306859250090feca86475fd0b9314878044723d92e0
     8fe8e19cad68e7afe169acaa5df3b44eff955647e6deff89fc96625255fcd4fa5ef2fdfb19e6
     c5000f5981fb398405d3e6d2abd1d1b3003333584e4988e2c5d52204dc1c4f2b250717885aea
     fc57ea5958dd6f553494bd9484bd5d62307098fa9ffab751751710e7070d1424b237fd2df4da
     5729d86d117053e424e04f74d7e2778277f76b11039693b0a089782eafa251a1bcc696176603
```

Bridge B_3 (found first). $y =$ 7b1a26bd4bd1605457e6.

```
P1 = 0d48d0dce4514fa0f21ef60586143d192a47979f1cc5c0b45754e3b2d5b5c7af710df9623cb9
     8902df6a6d009fab93622355cdcb2ce1f7e10a4f3334c9fe9b908f9efd63b81d1ae452070e3e
     7825b03e341b31218e0eacddbbcd641d215937e15f43942a1b70b2bfcf4777cd96047b367737
     6bfbe83c44e3bbefb3b5452703767c59314eb06290ac01e0882c877cf8634af052963f89d65d
     8183d43d0043ca339aa770c47edff5b6a0b123b6d1f1da7aaec073ba7fe2ad58f9f2047ee3e0
M  = 2c5c4f1bbcea58f4b077db08d5d131a4ccb9c76df49fe2be8b44bc178553fa65c1b8d4b3e06a
     ad5af252bc8ac12bca73602d801dbcf454f93dbdba167f260557ba6189b7238eb81ba62091b9
     bc1c934a22d57aef20a4615abf29316848f7854ae3b340e715758c0df3063fb1c80b01490394
     f0815c662d9100233adaa8c997826b61c1643c2d35214eaf000da17e3d5aaa5e3a1cd2776481
     a559004b27e2b7c52ef608d1ba027dd2dbec033cc18875235c3b87876135134e84598be6a4f9
P3 = 6ee7bdfba9954db114661a57b5a4cf6d82cb2a3ae3439a39ec8553eb90c1a005cb6183885954
     2147dc61f86229fee91769042b8c1f93a216d96e82d1f3a0c9c331cf5a5c52923c5ab5298e7d
     b70aca2eb29dfa2a50bf0fd12dc15923603cc0499a44883cd0879c5a1723ee290501dba5912a
     03fdc0f8a3ee4cffa65c83dc2e2349d5413f00394cbac1e307f70ac8acf82bc81d863e934edf
     f17cc560fede6070bd9aa926a9a64074f2ff11bb16e38c7bcc565785d773756f4f028697eac0
```

C Public halves of the one-sided challenges

Full 24-round Keccak-$f[1600]$; $x =$ `cef2e4428344e8e8ad398f` (Section 2.7.1). P_1 is given as rate bytes in FIPS 202 byte order (hex); the capacities x_{in} (`cap_in`) and x_{out} (`cap_out`) are hexadecimal little-endian integers. A completion is any (M, P_3) with $f\big(f(P_1\|x_{\text{in}}) \oplus (M\|0^c)\big) = (P_3\|x_{\text{out}})$; at $c = 92$ the block P_1 has $r = 1508$ bits, so the high four bits of its last byte are zero. The salted SHA3-256 commitments (Section 2.7.1; one opening backs both c88-B3 and c92-B3) are

```
c88-B3, c92-B3: 135259d6611d2be43a8b83fda6cad933a38ef54dd6ebbed6b343262cfe50a545
c88-B4:         2593472e5e2f119144820588a33dd9583809a2990bd8bee88878e71ade5fc52d
```

Challenge c88-B3 ($c = 88$). Claw label `f70a0908839e574478e139`; $c_{\text{eff}} = 92$ (general), 88 (loop).

```
cap_in  = cef2e4428344e8e8ad398f
cap_out = cef2e4428344e8e8ad398f
P1 = 98061a11e7b458426429b16098e16dab71cab59e82621f21987e18f50244df75d9781429ab3b
     31ddc16f0e60d8c60fcaf545d4745e8af5ba8c6a4691e1c967ff7be52351b8c7ab01405848af
     d3ef69cdba3273f1cc56d296b819f1e914b006c89c69b69b9ae50768e0d2cb8f7d5d7fe709e7
     9f0c64bc985346db21b36ef34cf8987e535c7d8092e23b6e9d36ae8bda69ab883bfe81f7527b
     335afe38ff6221188aa42a5395602842746532ba80eb2ec126655b6d381084992ff02d924b
```

Challenge c92-B3 ($c = 92$). Claw label `f70a0908839e574478e139`; $c_{\text{eff}} = 92$ (general), – (loop).

```
cap_in  = cef2e4428344e8e8ad398f4
cap_out = cef2e4428344e8e8ad398fa
P1 = 98061a11e7b458426429b16098e16dab71cab59e82621f21987e18f50244df75d9781429ab3b
     31ddc16f0e60d8c60fcaf545d4745e8af5ba8c6a4691e1c967ff7be52351b8c7ab01405848af
     d3ef69cdba3273f1cc56d296b819f1e914b006c89c69b69b9ae50768e0d2cb8f7d5d7fe709e7
     9f0c64bc985346db21b36ef34cf8987e535c7d8092e23b6e9d36ae8bda69ab883bfe81f7527b
     335afe38ff6221188aa42a5395602842746532ba80eb2ec126655b6d381084992ff02d920b
```

Challenge c88-B4 ($c = 88$). Claw label `990e297db33ae0cbf0ebdf`; $c_{\text{eff}} = 89$ (general), 88 (loop).

```
cap_in  = cef2e4428344e8e8ad398f
cap_out = cef2e4428344e8e8ad398f
P1 = 21077461eb8d5862ef037df7cd2399137aa22341450e1cab0f89857f6187e46504392213032d
     1c18094ca8158d7d5efd0603951c162ec3d21921350fe617e44c96b366f55944f5b7232fd5e3
     77258d0fb4235f06ecb2be7fc66670508ff9ad538d292f5b025233f573cd20c58e4213f844c4
     67f5fe48b5be9233f869b5029ebd3f1171c577078ea04466a53ed88237979fd6b9144f5b7115
     4838aee555944aff5d17f28ce2684b637931ef4f5ab387cbade325386a43f549df22e16d73
```

D Test vectors

Examples for Proposition 3 and Theorem 1. The full-round $f[200]$ and $f[1600]$ ($c = 32$) instances of Remark 6, including the connectors and the test message, are listed in `02_bridges_fig1/general_bridge_examples/vectors.md` and are re-checked by `verify.py` in about one second. The $f[1600]$ single-state bridge has $x =$ `e2a3db51` and the $f[1600]$ loop has $x =$ `e7f81f8a`.

Keccak-$f[1600]$, 24 rounds, $r = 1568$, $c = 32$; hex shortened as `first16B..last8B`; full values are in `test_vectors.json`.

Bridge 0 (seed 1). $f(P_1\|0) = (P_2\|C)$, $f^{-1}(P_3\|0) = (P_4\|C)$, $C =$ `765aaf3a`, message blocks P_1, $P_2 \oplus P_4$, P_3.

```
P1 = 49ca185bfa62d2ce5bdd215f622cfdc4..1ac1f2bf7f051442
P2 = 0d8995bd5a69bcf1df91e3d194f49e0f..fba19d62002aafb3
P3 = 42618b015a5509034f1532476ea8615e..c29ccac456c78fb0
P4 = 50e49ae5e7f11b436c5fb0089756d0fd..636183360804bedd
```

Second-preimage pair (Keccak pad `0x01`, bridge 0, 256-bit output).

```
M  (196 B) = 1e5180f383a5dcf31ae239e5999f8e6b..ce6f8ea9e808d44a
M2 (588 B) = 49ca185bfa62d2ce5bdd215f622cfdc4..0cf3446dbecf5bfa
digest     = 49b62b94922efeaa0f397854dcc09236
             a5750f828ad227e08577f1e3a00f13a0
```

Multicollision $k = 8$ (256 messages of 5880 bytes, `multicollision.messages` in the JSON).

```
Keccak pad: dd1237f346f5a2472ff5c8d17a14c99d1c6f5c325993d1a610c9b679437955f1
SHA-3 pad:  31366f636ce8d0a83bd4f525b2ffd173f128749c5e61a4d155b8630670403a45
```

Reproduce. In `02_bridges_fig1/f1600_c32/` run

```
python3 gen.py && python3 verify_vectors.py
```

(`gen.py` needs `01_keccak_reference/keccak/cfast.py`; the checker is standalone).

E Experimental results

All experiments use Keccak-$f[200]$ ($w = 8$), first R rounds, and every reported bridge was re-verified by recomputation. Statements in this section are empirical unless stated otherwise.

E.1 Literal bridge vs. birthday connector search, $c \in \{16, 20, 24\}$

540 runs ($R = 1..18$, 5 starts, both methods), all ended in a verified result (a single-state bridge for the literal method, a two-point connector for the birthday method, Section 2.6); no cycles; no birthday hit was literal ($i = j$). Wall time 53.5 min on 6 cores. Ratios of means

Table 8: Literal single-state bridge search vs. birthday connector search (Algorithm 2), $f[200]$, 90 runs per row. Norm $= 2^c$ (literal), $2^{c/2}$ (birthday). The two methods find different objects; fixed-capacity MITM bridges are measured in Section 2.6.

c	method	mean k	median k	mean/norm	median/norm	expected mean/norm
16	literal	$7.727 \cdot 10^4$	$6.096 \cdot 10^4$	1.179	0.930	1.000
16	birthday	220.2	200.5	0.860	0.783	0.886
20	literal	$1.049 \cdot 10^6$	$7.859 \cdot 10^5$	1.000	0.749	1.000
20	birthday	1002	920	0.979	0.898	0.886
24	literal	$2.23 \cdot 10^7$	$1.842 \cdot 10^7$	1.329	1.098	1.000
24	birthday	3586	3379	0.876	0.825	0.886

literal/birthday: 350.9, 1046.2, 6218.2 (theory 288.9, 1155.5, 4621.8). Since the methods find different objects, these ratios check the 2^c and $2^{c/2}$ laws; they are not a speed-up for the same task. Kruskal–Wallis over R gives one small value, $p = 0.006$ ($c = 24$, literal), which remains below 0.05 after Bonferroni over the 6 tests (0.036); with 5 samples per cell the asymptotic p-value is only approximate, and all OLS slopes are non-significant. We did not detect any other R dependence; a permutation test and more starts for this cell are left open.

E.2 Birthday connector search, $c = 32$

On an Apple Silicon M5 Max Pro laptop, 90 runs ($c = 32$, $R = 1..18$, 5 starts) all found a connector that verified; mean $k/2^{c/2} = 0.86$ (theory 0.89), per-R means 0.64–1.10 ($n = 5$). As in Table 8 these are connectors, not bridges (Section 2.6). Verified bridges in the sense of Definition 2 come from the fixed-capacity MITM search ($c \leq 24$, Section 2.6), the full-round $c = 32$ examples (Remark 6, Section E.4) and the distinguished-point search ($c \leq 48$).

E.3 Capacity-map functional graphs

For the inverse map $h'_R(x) = \mathrm{cap}\, f_R^{-1}(0^r \| x)$ and forward h_R, we computed exhaustive functional graphs and walk statistics.

- Exhaustive $c = 32$, $R = 2$ inverse (rate seed 0): giant component with cycle length 286, basin $0.998151N$; 9 components (E 11.73); image $0.632116N$ ($\approx 1 - 1/e$); 1534 cyclic nodes vs. E 82,137 (single-map flag, $p = 5.5 \cdot 10^{-4}$).
- The 286-cycle is a property of one map: 0/180 perturbed h'_2 maps had main cycle ≤ 286 (random expectation 0.55%).
- Fixed-rate sweep $c = 32$, $R = 2$: 78,394 maps; BH over 76 tests, min $q = 0.164$; nothing significant.
- Exhaustive $c = 28$, $R = 1..18$, both directions: for $R \geq 2$, nothing survives BH over 340 tests (min $q = 0.422$); image fractions 0.6321 ± 0.0001. $R = 1$ is degenerate (as expected).
- Outlier census over all Keccak maps (266): min $q = 0.13$, Bonferroni global $p = 0.18$ [8].

The 286-cycle as a message mode. The 286-cycle of $h'_2(x) = \mathrm{cap}\, f_2^{-1}(0^{168} \| x)$ ($c = 32$, $R = 2$, Keccak-f[200]) extends the observation of Fig. 1 from a single return to the zero state to a repeatable loop. Write the cycle as $x_0, \ldots, x_{285}$ and $f_2^{-1}(0^{168} \| x_i) = y_i \| x_{i+1}$. In the forward (hashing) sponge, absorbing y_{i-1} into the state $0^{168} \| x_i$ gives $f_2(y_{i-1} \| x_i) = 0^{168} \| x_{i-1}$, so 286 blocks L walk the cycle and return to exactly the same 200-bit state; for a state $r \| x_i$ the block $r \oplus y_{i-1}$ does the same, since the block controls the rate fully. Hence a prefix P that reaches a cycle capacity gives $P \| L^k \| S$ with the same digest for every $k \geq 0$ and every suffix S: an unbounded family of colliding messages of different lengths at no extra cost. The tail of h'_2 from $x = 0$ ($\mu = 73{,}830$, entering at x_{161}) read in the forward direction is a 73,830-block message T from the cycle state $0^{168} \| x_{161}$ to 0^{200}, the IV, so $P_{\mathrm{enter}} \| L^k \| T$ is a neutral prefix that returns the sponge to the IV (checked by recomputation). Entering the cycle from the IV is generic: one block hits one of 286 capacities with probability $286/2^{32}$, about $2^{23.8}$ calls of f_2, and in general the cost is $\approx 2^c/\ell$ for a cycle of length ℓ. The effect is therefore not a weakness of Keccak but a consequence of $c = 32$ being small; the 286-cycle itself is a lucky instance (per-map $p = 6.7 \cdot 10^{-4}$, global $p = 0.18$ after the look-elsewhere correction above).

Full campaign (Apple Silicon M5 Max Pro, 10 threads, 1987 min wall, finished 25.10.2026 02:07 Oslo; all 72 jobs exit code 0, all self-tests passed, all 36 $c = 32$ graphs and all $18 \times 78{,}394$ sweep maps present). Tables 9 and 10 give the per-cell results. Each exhaustive cell costs 2^{32} map evaluations (table 50–1800 s, total 1000–2700 s per cell for $R \geq 2$); each R of the sweep costs about $2^{36.0}$ map evaluations. Benjamini–Hochberg over the 340 exhaustive $c = 32$ tests for $R \geq 2$ gives min $q = 0.0032$, coming only from the already known $R = 2$ inverse map (giant cycle 286, see above); without that map, min $q = 0.137$ (smallest raw $p = 4.1 \cdot 10^{-4}$: $R = 15$ inverse, giant cycle 1888 on a 0.59 basin). Over the 136 sweep count tests for $R \geq 2$, min $q = 0.177$. Summed over $R = 2..18$ and both directions, the number of maps with main cycle ≤ 1000 is above the random-mapping prediction: 19,673 vs. 19,171 expected of 1,332,664 non-zero-rate maps (+2.6%, $\approx 3.6\sigma$ under a Poisson approximation). This excess is spread

Table 9: Full $c = 32$ exhaustive campaign, $f[200]$, first R rounds, zero rate, both directions ($N = 2^{32}$; E[components]= 11.73, E[cyclic nodes]= 82,137, image $1 - 1/e = 0.63212$). Min p = smallest single-test p of 10 random-mapping tests (uncorrected). $R = 1$ is degenerate.

R	dir	comp.	giant cycle	giant share	cyclic/E	image	min p
1	i	12	6811	0.9788	0.103	0.10011	$< 10^{-15}$
1	f	59	164240	0.5608	3.723	0.59626	$< 10^{-15}$
2	i	9	286	0.9982	0.019	0.63212	$9.5 \cdot 10^{-6}$
2	f	13	25146	0.7456	1.112	0.63212	0.071
3	i	13	28775	0.8087	0.790	0.63211	0.038
3	f	7	16483	0.7092	0.789	0.63212	0.031
4	i	15	52418	0.6682	1.020	0.63212	0.16
4	f	19	46343	0.4571	1.224	0.63213	0.052
5	i	11	126361	0.7269	1.733	0.63212	0.047
5	f	8	61598	0.9237	0.818	0.63211	0.1
6	i	12	32081	0.5857	0.735	0.63213	0.11
6	f	12	36191	0.8583	0.661	0.63213	0.02
7	i	11	74413	0.7657	1.261	0.63213	0.05
7	f	18	59964	0.6111	1.557	0.63213	0.11
8	i	8	136168	0.9904	1.790	0.63212	0.074
8	f	18	12560	0.7692	0.707	0.63212	0.018
9	i	16	73358	0.5534	1.600	0.63212	0.17
9	f	8	32400	0.6204	1.299	0.63212	0.12
10	i	12	33519	0.4838	1.441	0.63212	0.12
10	f	8	95574	0.9516	1.444	0.63212	0.19
11	i	10	83354	0.8854	1.609	0.63213	0.093
11	f	8	19577	0.9998	0.272	0.63212	0.0031
12	i	8	7910	0.9660	0.273	0.63212	0.0073
12	f	12	73717	0.9864	1.122	0.63212	0.087
13	i	10	10519	0.6251	0.685	0.63212	0.013
13	f	13	69514	0.5871	1.824	0.63212	0.15
14	i	8	61850	0.9976	0.772	0.63212	0.032
14	f	14	43816	0.7696	0.951	0.63212	0.2
15	i	14	1888	0.5885	0.279	0.63212	$4.1 \cdot 10^{-4}$
15	f	14	10429	0.6177	0.858	0.63212	0.013
16	i	5	64436	0.9999	0.792	0.63213	0.008
16	f	12	18560	0.7474	1.134	0.63212	0.039
17	i	8	41305	0.9958	0.536	0.63212	0.031
17	f	9	46024	0.9472	0.968	0.63212	0.12
18	i	11	50557	0.6209	1.800	0.63212	0.056
18	f	11	8274	0.4642	0.674	0.63212	0.0079

evenly over R and directions (no single cell is significant after correction), and the "main" cycle is the most-hit cycle of 16 walks, while the prediction $F(1000)$ assumes the exact giant cycle; we therefore attribute it to this approximation rather than to Keccak structure. A calibration confirms this: the same sweep code (16 walks per map, same starts and early exit) run on 300,000 random functions on 32 bits (keyed splitmix64) gives a main cycle ≤ 1000 in 4,367 maps, a rate of $1.456\% \pm 0.022\%$ against $F(1000) = 1.439\%$. With this empirical rate the expectation becomes 19,399, and the Keccak excess shrinks to +1.4%, about 0.8σ including the calibration error, so it is consistent with a random mapping. (p-values are asymptotic, tests overlap.) $R = 1$ is again degenerate (image $0.100N$ inverse, $0.596N$ forward).

A direction for separate research: algebraic bridges in reduced variants. For $R = 2$, one-sided capacity preimages with linear structures (row-4 χ^{-1} gives $5w$ linear equations without guesses) found 18/18 verified bridges at lane width $w = 16$ with $\log_2 W = 43.95$ work units (prediction $3w - 2k = 44$, $k = 2$; 1793.8 CPU-hours), and the growth from $w = 8$ to $w = 16$ is below the generic birthday rate. (two widths only; work units are not exactly permutation

Table 10: Fixed-rate walk sweeps, $c = 32$, per R and direction: 39,197 maps each (zero, 168 HW-1, 14,028 HW-2, 5000 HW-3, 20,000 random rates; 16 walks per map). "≤1000" = maps whose main cycle has length ≤ 1000, observed (expected under the random-mapping law); p = smallest binomial upper-tail p over the four rate sets (uncorrected).

	inverse h'_R			forward h_R			
R	min main	≤1000 (exp)	min p	min main	≤1000 (exp)	min p	$\log_2$ evals
1	1	2942 (564)	$< 10^{-15}$	1	734 (564)	$< 10^{-15}$	35.21
2	3	585 (564)	0.056	3	596 (564)	0.027	36.01
3	1	574 (564)	0.31	2	583 (564)	0.11	36.01
4	1	576 (564)	0.22	1	578 (564)	0.13	36.01
5	3	580 (564)	0.097	2	607 (564)	0.042	36.01
6	1	565 (564)	0.3	4	567 (564)	0.41	36.01
7	4	583 (564)	0.13	5	609 (564)	0.011	36.00
8	2	550 (564)	0.27	2	552 (564)	0.44	36.01
9	1	572 (564)	0.24	2	569 (564)	0.31	36.01
10	3	575 (564)	0.18	5	589 (564)	0.16	36.01
11	2	594 (564)	0.042	6	575 (564)	0.094	36.01
12	6	603 (564)	0.15	1	610 (564)	0.051	36.01
13	3	564 (564)	0.42	2	580 (564)	0.15	36.01
14	1	554 (564)	0.36	3	590 (564)	0.083	36.01
15	3	585 (564)	0.0026	1	573 (564)	0.25	36.01
16	2	540 (564)	0.37	6	555 (564)	0.22	36.01
17	7	573 (564)	0.37	4	598 (564)	0.0013	36.01
18	4	607 (564)	0.051	2	563 (564)	0.26	36.00

calls). This is an anomaly search in reduced-round variants, not needed for the protocols of this paper; we leave its baseline, a $w = 32$ point and the $R = 3$ case to future work.

E.4 Concrete instances on the full Keccak-$f[1600]$ with $c = 32$

Setup. We instantiate the constructions of Section 2 on the full 24-round Keccak-$f[1600]$ with rate $r = 1568$ bits (196 bytes) and capacity $c = 32$, the capacity being the top 32 bits of lane 24 (state bytes 196..199). We hash with both the Keccak padding (domain byte `0x01`) and the SHA-3 padding (`0x06`), with 256-bit outputs. f and f^{-1} come from the repository's C code; all results were generated with fixed seeds by `gen.py` (111 s in total on the box).

Bridges. Eight birthday runs, each with 2^{16} f and 2^{16} f^{-1} calls (6.6–6.8 s per run), gave 11 bridges in total (one or two per run).

Second preimages. With two bridges, each of three messages ($|M| = 196,203,442$ bytes) under both paddings gets a second preimage $M_2 = B_1 \| B_2 \| (P_3 \oplus M[0{:}196]) \| M[196{:}]$: 12 verified pairs.

Multicollision. One inner collision from the zero state (79,073 f calls, 1.6 s) and one bridge from the collision state give a $k = 8$ multicollision: 256 distinct messages of 5880 bytes with equal digests under both paddings. Generator total, including hashing verification: 160,486 f and 65,536 f^{-1} calls.

Preimages. The generic sponge meet-in-the-middle needs $\approx 2^{c/2+1} = 2^{17}$ permutation calls for any output length $n \leq r$, against 2^n for brute force.

Independent verification. The checker `verify_vectors.py` uses only the unmodified XKCP `CompactFIPS202.py` [49] (SHA-256 pinned). It first passes the official KATs for SHA3-256 and Keccak-256, cross-checks against `hashlib` and pycryptodome, and then checks every bridge, pair, multicollision message and preimage (6.4 s, `ALL PASS`).

Table 11: Full Keccak-$f[1600]$, 24 rounds, $c = 32$: measured costs (`gen.log`).

Object	f calls	f^{-1} calls	wall s	generic reference
Bridge run (8 runs, 11 bridges)	65,538–65,540	65,536	6.6–6.8	$\approx 2^{c/2}$ each side
Second preimages (12 pairs)	reuse bridges 0 and 1			–
Multicollision $k = 8$ (256 msgs)	160,486	65,536	≈9.6	Joux: $8 \times 2^{c/2}$
Preimage $n = 32$, SHA-3 pad	90,471	180,936	15.8	2^{17} vs. 2^{32}
Preimage $n = 32$, Keccak pad	48,480	96,954	8.5	2^{17} vs. 2^{32}
Preimage $n = 256$, SHA-3 pad	134,758	269,510	23.6	2^{17} vs. 2^{256}

All costs above are generic in c: they are birthday or meet-in-the-middle searches on a 32-bit capacity and exploit *no* property of Keccak-f. They are consistent with the non-claim: with $c = 512$ the same procedures cost $\approx 2^{256}$.

Towards larger capacities. Instances with $c = 64$ and $c = 80$ ($\approx 2^{32}$ resp. 2^{40} calls per bridge with the birthday search, with distinguished points [73] to bound memory) are run with the distinguished-point code above (`02_bridges_fig1/tmto_dp/pcs.c`; its multithreaded version also runs on $f[1600]$). A $c = 80$ bridge found this way is given in Remark 7.

References

[1] Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John Schanck. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In *Selected Areas in Cryptography – SAC 2016*, volume 10532 of *LNCS*, pages 317–337. Springer, 2017. Cryptology ePrint Archive, Paper 2016/992.

[2] Elena Andreeva, Joan Daemen, Bart Mennink, and Gilles Van Assche. Security of keyed sponge constructions using a modular proof approach. In *Fast Software Encryption (FSE 2015)*, volume 9054 of *LNCS*, pages 364–384. Springer, 2015.

[3] Jean-Philippe Aumasson, Eric Brier, Willi Meier, María Naya-Plasencia, and Thomas Peyrin. Inside the hypercube. In *ACISP 2009*, volume 5594 of *LNCS*, pages 202–213, 2009.

[4] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and María Naya-Plasencia. Quark: A lightweight hash. *Journal of Cryptology*, 26(2):313–339, 2013.

[5] Carsten Baum, Marvin Beckmann, Ward Beullens, Shibam Mukherjee, and Christian Rechberger. Concretely efficient blind signatures based on VOLE-in-the-head proofs and the MAYO trapdoor. In *35th USENIX Security Symposium (USENIX Security 26)*, pages 6325–6344, Baltimore, MD, USA, August 2026. USENIX Association. Full version: Cryptology ePrint Archive, Paper 2026/109.

[6] Carsten Baum, Ward Beullens, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Kloß, Christian Majenz, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. FAEST v3: Algorithm specifications, version 3.0. Submission to the NIST call for additional post-quantum signature schemes, Round 3, https://faest.info/faest-spec-v3.0.pdf, September 2026.

[7] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Kloß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In *Advances in Cryptology – CRYPTO 2023*, volume 14085 of *LNCS*, pages 581–615. Springer, 2023. Cryptology ePrint Archive, Paper 2023/996.

[8] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. J. Royal Statistical Society B 57(1):289–300, 1995.

[9] Daniel J. Bernstein. Second preimages for 6 (7? (8??)) rounds of Keccak? Post to the NIST hash-forum mailing list, 27 November 2010; archived at https://ehash.isec.tugraz.at/uploads/6/65/NIST-mailing-list_Bernstein-Daemen.txt, 2010.

[10] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. The Keccak crunchy crypto collision and pre-image contest. https://keccak.team/crunchy_contest.html. Accessed October 2026.

[11] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Farfalle: parallel permutation-based cryptography. *IACR Transactions on Symmetric Cryptology*, 2017(4):1–38, 2017.

[12] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic sponge functions. Version 0.1, STMicroelectronics and NXP Semiconductors, January 2011.

[13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In *Selected Areas in Cryptography (SAC 2011)*, volume 7118 of *LNCS*, pages 320–337. Springer, 2012.

[14] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. CAESAR submission: Ketje v2. https://keccak.team/files/Ketjev2-doc2.0.pdf, 2016. Document version 2.0.

[15] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. CAESAR submission: Keyak v2. https://keccak.team/files/Keyakv2-doc2.2.pdf, 2016. Document version 2.2.

[16] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, Ronny Van Keer, and Benoît Viguier. KangarooTwelve: Fast hashing based on Keccak-*p*. In *Applied Cryptography and Network Security (ACNS 2018)*, volume 10892 of *LNCS*, pages 400–418. Springer, 2018.

[17] Alex Biryukov and Dmitry Khovratovich. Equihash: Asymmetric proof-of-work based on the generalized birthday problem. In *NDSS 2016*, 2016.

[18] Blockchain.com. Total hash rate (th/s) of the Bitcoin network. https://www.blockchain.com/explorer/charts/hash-rate, 2026. 30-day average 1,008 EH/s on 23 October 2026. Accessed 25 October 2026.

[19] Bloomberg Law. Elon musk aims to double colossus 2's nvidia chips by year-end. https://news.bloomberglaw.com/artificial-intelligence/elon-musk-aims-to-double-colossus-2s-nvidia-chips-by-year-end, 2026. Accessed 25 October 2026.

[20] Richard P. Brent. An improved Monte Carlo factorization algorithm. *BIT*, 20(2):176–184, 1980.

[21] Vitalik Buterin and Christian Reitwiessner. EIP-197: Precompiled contracts for optimal ate pairing check on the elliptic curve alt_bn128. https://eips.ethereum.org/EIPS/eip-197, 2017. Ethereum Improvement Proposal, status: Final.

[22] Antonio Salazar Cardozo and Zachary Williamson. EIP-1108: Reduce alt_bn128 precompile gas costs. https://eips.ethereum.org/EIPS/eip-1108, 2018. Ethereum Improvement Proposal, status: Final.

[23] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM CCS 2017*, pages 1825–1842, 2017.

[24] Greg Colvin, Andrea Lanfranchi, Michael Carter, and IfDefElse. EIP-1057: ProgPoW, a programmatic proof-of-work. https://eips.ethereum.org/EIPS/eip-1057, 2018. Ethereum Improvement Proposal, status: Stagnant.

[25] Joan Daemen, Bart Mennink, and Gilles Van Assche. Full-state keyed duplex with built-in multi-user support. In *Advances in Cryptology – ASIACRYPT 2017*, volume 10625 of *LNCS*, pages 606–637. Springer, 2017.

[26] Itai Dinur. Cryptanalytic applications of the polynomial method for solving multivariate equation systems over GF(2). In *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 374–403. Springer, 2021. ePrint 2021/578.

[27] Itai Dinur, Orr Dunkelman, and Adi Shamir. New attacks on Keccak-224 and Keccak-256. In *Fast Software Encryption – FSE 2012*, volume 7549 of *LNCS*, pages 442–461. Springer, 2012. Journal version: Improved Practical Attacks on Round-Reduced Keccak, J. Cryptology 27(2):183–209, 2014, doi:10.1007/s00145-012-9142-5.

[28] Itai Dinur, Orr Dunkelman, and Adi Shamir. Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials. In *Fast Software Encryption – FSE 2013*, volume 8424 of *LNCS*, pages 219–240. Springer, 2014. ePrint 2012/672.

[29] Niels Ferguson, Stefan Lucks, and Kerry A. McKay. Symmetric states and their structure: Improved analysis of CubeHash. Cryptology ePrint Archive, Report 2010/273, 2010.

[30] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019.

[31] Koustabh Ghosh. Multiplication-based primitive design and differential probability of trails. PhD thesis (public defence), Radboud University, Nijmegen, 2026. Public defence on 5 October 2026; promotor: J. Daemen. https://www.ru.nl/en/about-us/events/multiplication-based-primitive-design-and-differential-probability-of-trails.

[32] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for boolean circuits. In *25th USENIX Security Symposium*, pages 1069–1083, 2016. Key GMO2016 in our notes; checked 2026-10-25 on usenix.org.

[33] Danilo Gligoroski. Beyond the event horizon of Keccak: code, test vectors and Lean 4 proofs. Zenodo, 2026. Version 1.7, 12 November 2026, doi:10.5281/zenodo.24163450. Zenodo, concept DOI 10.5281/zenodo.23727976 (all versions); version 1.6.1: doi:10.5281/zenodo.23918542.

[34] Danilo Gligoroski, Rune Steinsmo Ødegård, and Rune Erlend Jensen. OBSERVATION: An explicit form for a class of second preimages for any message M for the SHA-3 candidate Keccak. Cryptology ePrint Archive, Report 2011/261, 2011.

[35] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT 2016*, volume 9666 of *LNCS*, pages 305–326. Springer, 2016.

[36] Jian Guo, Guohong Liao, Guozhen Liu, Meicheng Liu, Kexin Qiao, and Ling Song. Practical collision attacks against round-reduced SHA-3. *Journal of Cryptology*, 33(1):228–270, 2020. ePrint 2019/147.

[37] Jian Guo, Guozhen Liu, Ling Song, and Yi Tu. Exploring SAT for cryptanalysis: (quantum) collision attacks against 6-round SHA-3. In *Advances in Cryptology – ASIACRYPT 2022, Part III*, volume 13793 of *LNCS*, pages 645–674. Springer, 2022. ePrint 2022/184.

[38] Jian Guo, Meicheng Liu, and Ling Song. Linear structures: Applications to cryptanalysis of round-reduced Keccak. In *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 249–274. Springer, 2016. ePrint 2016/878.

[39] Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON family of lightweight hash functions. In *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *LNCS*, pages 222–239. Springer, 2011.

[40] Le He, Xiaoen Lin, and Hongbo Yu. Improved preimage attacks on 4-round Keccak-224/256. *IACR Transactions on Symmetric Cryptology*, 2021(1):217–238, 2021.

[41] Senyang Huang, Orna Agmon Ben-Yehuda, Orr Dunkelman, and Alexander Maximov. Finding collisions against 4-round SHA3-384 in practical time. *IACR Transactions on Symmetric Cryptology*, 2022(3):239–270, 2022.

[42] A. Huelsing, D. Butin, S. Gazdag, J. Rijneveld, and A. Mohaisen. XMSS: extended Merkle signature scheme. RFC 8391, May 2018.

[43] International Association for Cryptologic Research. IACR policy on the use of AI. https://iacr.org/policy/AIPolicy.html, 2026. Revised 22 September 2026; accessed 25 October 2026.

[44] Irreducible. Binius benchmarks (binius64 vs. plonky3, Keccak-256). https://www.binius.xyz/benchmarks, 2026. Accessed 25 October 2026.

[45] Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. Implementing grover oracles for quantum key search on AES and LowMC. In *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 280–310. Springer, 2020. Full version: Cryptology ePrint Archive, Paper 2019/1146.

[46] Antoine Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In *CRYPTO 2004*, volume 3152 of *LNCS*, pages 306–316, 2004.

[47] Philipp Jovanovic, Atul Luykx, Bart Mennink, Yu Sasaki, and Kan Yasuda. Beyond conventional security in sponge-based authenticated encryption modes. *Journal of Cryptology*, 32(3):895–940, 2019.

[48] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *ACM CCS 2018*, pages 525–537, 2018.

[49] Keccak Team. eXtended Keccak Code Package (XKCP), standalone/compactfips202/python. https://github.com/XKCP/XKCP. Accessed October 2026.

[50] John Kelsey and Tadayoshi Kohno. Herding hash functions and the Nostradamus attack. In *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 183–200, 2006.

[51] John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 474–490, 2005.

[52] Charlotte Lefevre and Bart Mennink. Tight preimage resistance of the sponge construction. In *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 185–204. Springer, 2022. ePrint 2022/734.

[53] Ting Li and Yao Sun. Preimage attacks on round-reduced Keccak-224/256 via an allocating approach. In *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 556–584. Springer, 2019. ePrint 2019/248.

[54] Ting Li, Yao Sun, Maodong Liao, and Dingkang Wang. Preimage attacks on the round-reduced Keccak with cross-linear structures. *IACR Transactions on Symmetric Cryptology*, 2017(4):39–57, 2017.

[55] Xiaoen Lin, Le He, Zhengrong Lu, Yantian Shen, Chongxu Ren, and Hongbo Yu. Internal differential structure: Preimage attacks on up to 5-round Keccak. *Designs, Codes and Cryptography*, 93(9):3761–3808, 2025.

[56] Xiaoen Lin, Le He, and Hongbo Yu. Improved preimage attacks on 3-round Keccak-224/256. *IACR Transactions on Symmetric Cryptology*, 2021(3):84–101, 2021.

[57] Xiaoen Lin, Le He, and Hongbo Yu. Practical preimage attacks on 3-round Keccak-256 and 4-round Keccak[r=640, c=160]. *IACR Transactions on Symmetric Cryptology*, 2025(1):328–356, 2025. ePrint 2023/101.

[58] Xiaoen Lin, Zhengrong Lu, Yantian Shen, Chongxu Ren, Hongbo Yu, Enming Dong, and Wenhao Wu. Practical preimage attack on 5-round Keccak[r=640, c=160]. *IACR Transactions on Symmetric Cryptology*, 2026(1):189–211, 2026.

[59] Fukang Liu, Takanori Isobe, Willi Meier, and Zhonghao Yang. Algebraic attacks on round-reduced Keccak. In *Information Security and Privacy – ACISP 2021*, volume 13083 of *LNCS*, pages 91–110. Springer, 2021. ePrint 2020/346.

[60] D. McGrew, M. Curcio, and S. Fluhrer. Leighton-Micali hash-based signatures. RFC 8554, April 2019.

[61] Paweł Morawiecki, Josef Pieprzyk, and Marian Srebrny. Rotational cryptanalysis of round-reduced Keccak. In *Fast Software Encryption – FSE 2013*, volume 8424 of *LNCS*, pages 241–262. Springer, 2014. ePrint 2012/546.

[62] National Institute of Standards and Technology. SHA-3 standard: Permutation-based hash and extendable-output functions. Technical Report FIPS PUB 202, NIST, 2015.

[63] National Institute of Standards and Technology. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf, December 2016.

[64] Polygon Zero. Plonky3: a toolkit for polynomial IOPs. https://github.com/Plonky3/Plonky3, version 0.8.0, 2026.

[65] Kexin Qiao, Ling Song, Meicheng Liu, and Jian Guo. New collision attacks on round-reduced Keccak. In *Advances in Cryptology – EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 216–243. Springer, 2017. ePrint 2017/128.

[66] Lingyue Qin, Jialiang Hua, Xiaoyang Dong, Hailun Yan, and Xiaoyun Wang. Meet-in-the-middle preimage attacks on sponge-based hashing. In *Advances in Cryptology – EUROCRYPT 2023, Part IV*, volume 14007 of *LNCS*, pages 158–188. Springer, 2023. ePrint 2022/1714.

[67] Mahesh Sreekumar Rajasree. Cryptanalysis of round-reduced KECCAK using non-linear structures. In *Progress in Cryptology – INDOCRYPT 2019*, volume 11898 of *LNCS*, pages 175–192. Springer, 2019. ePrint 2019/884.

[68] Christian Reitwiessner. EIP-196: Precompiled contracts for addition and scalar multiplication on the elliptic curve alt_bn128. https://eips.ethereum.org/EIPS/eip-196, 2017. Ethereum Improvement Proposal, status: Final.

[69] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical Report MIT/LCS/TR-684, MIT, 1996.

[70] Ling Song, Guohong Liao, and Jian Guo. Non-full Sbox linearization: Applications to collision attacks on round-reduced Keccak. In *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 428–451. Springer, 2017. ePrint 2017/529.

[71] Siwei Sun, Shun Li, Zhiyu Zhang, Charlotte Lefevre, Bart Mennink, Zhen Qin, and Dengguo Feng. Permutation-based hashing with stronger (second) preimage resistance. In *Advances in Cryptology – CRYPTO 2026, Part VI*, volume 16805 of *LNCS*, pages 131–162. Springer, 2026. ePrint 2025/963.

[72] Yi Tu, Ling Song, Huaijin Wu, Jian Guo, Jian Weng, and Chaoping Xing. Enhancing SAT solving to find (near) collisions in 6-round SHA-3 variants. Cryptology ePrint Archive, Paper 2026/2107, 2026.

[73] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999.

[74] Benoît Viguier, David Wong, Gilles Van Assche, Quynh Dang, and Joan Daemen. KangarooTwelve and TurboSHAKE. RFC 9861, IRTF, October 2025.

[75] Christof Zalka. Grover's quantum searching algorithm is optimal. *Physical Review A*, 60(4):2746–2751, 1999. arXiv:quant-ph/9711070.

[76] Zhongyi Zhang, Chengan Hou, and Meicheng Liu. Collision attacks on round-reduced SHA-3 using conditional internal differentials. In *Advances in Cryptology – EUROCRYPT 2023, Part IV*, volume 14007 of *LNCS*, pages 220–251. Springer, 2023.

[77] Zhongyi Zhang, Chengan Hou, and Meicheng Liu. Probabilistic linearization: Internal differential collisions in up to 6 rounds of SHA-3. In *Advances in Cryptology – CRYPTO 2024, Part IV*, volume 14923 of *LNCS*, pages 241–272. Springer, 2024. ePrint 2024/1136.

[78] Zhongyi Zhang, Chengan Hou, and Meicheng Liu. Preimage attacks on up to 5 rounds of SHA-3 using internal differentials. In *Advances in Cryptology – EUROCRYPT 2025, Part I*, volume 15601 of *LNCS*, pages 333–363. Springer, 2025. ePrint 2025/439.