CATENA Variants

Different Instantiations for an Extremely Flexible Password-Hashing Framework

Stefan Lucks Jakob Wenzel

Bauhaus-Universität Weimar, Germany

December 7

Passwords 2015
Cambridge (UK)

Outline

Section 1

Motivation

Motivation

- Password Hashing Competition (PHC, 2013 - 2015) winner
  - Argon2 (Argon2i, Argon2d, Argon2di)
- PHC special recognitions
  - CATENA
  - Lyra2
  - MAKWA
  - yescrypt
- heeded ideas and possible threats
  - memory-hardness / low-memory attacks
  - garbage-collector / cache-timing attacks
  - password-(in)dependent memory-accesses
  - FPGA/ASIC/GPU resistance
  - attacks on iterated compression functions
  - delegation feature
  - client-independent update / server relief
  - ...

Motivation (cont'd)

Why are we looking further at CATENA?

- agile framework (published already in 2012)
- pioneer regarding to flexible memory-demanding password hashing
- first password-hashing scheme which intentionally provides client-independent updates **and** server relief
- provides cache-timing resistance (depending on instantiation)

Flexibility of CATENA

- (cryptographic) hash function H and H'
- memory-hard function F (BRG$^g_\lambda$, SBRG$^g_\lambda$, GR2$^g_\lambda$, GR3$^g_\lambda$, DBG$^g_\lambda$)
- time- and memory-cost parameter (λ, g)
- extensions to $flap$ (Γ, Φ)

Security Considerations

- common security goals for memory-demanding password hashing
  - preimage security
  - resistance against low-memory attacks
  - FPGA/ASIC/GPU resistance
- application-depending security goals
  - resistance against (weak) garbage-collector attacks
  - resistance against cache-timing attacks
- additional requirement for key derivation
  - random-oracle security

Section 2

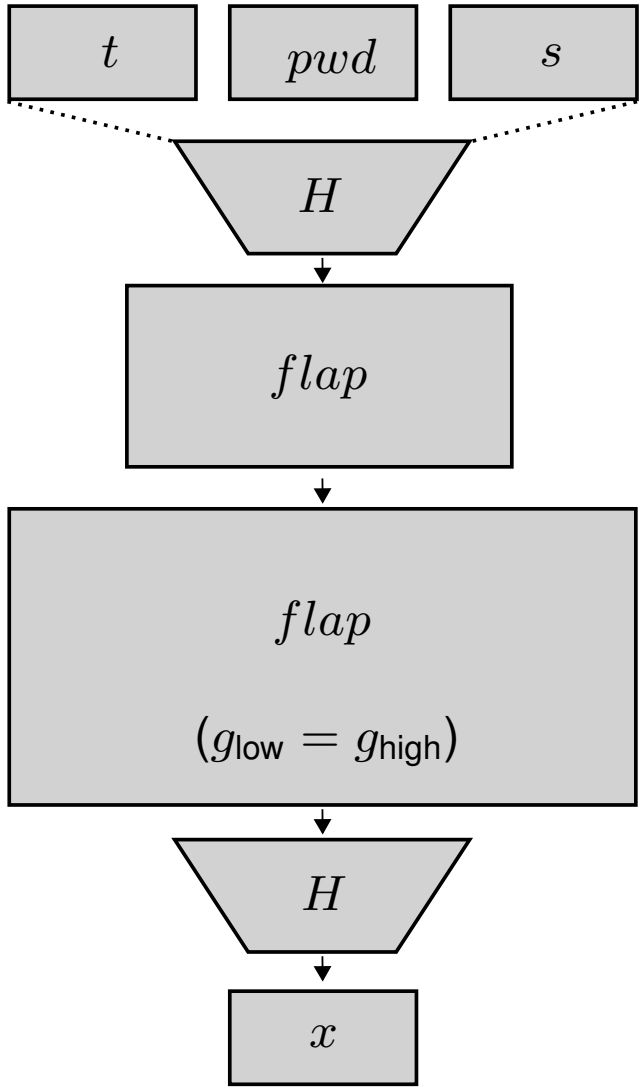
CATENA

CATENA – The Framework

Algorithm 1 CATENA

1: $x \leftarrow H(t \mid\mid pwd \mid\mid s)$
2: $x \leftarrow flap(\lceil g_{\text{low}}/2 \rceil, x, \gamma)$
3: **for** $g = g_{\text{low}}, \ldots, g_{\text{high}}$ **do**
4: $\quad x \leftarrow flap(g, x \mid\mid 0^*, \gamma)$
5: $\quad x \leftarrow H(g \mid\mid x)$
6: $\quad x \leftarrow truncate(x, m)$
7: **end for**
8: **return** x

t	tweak	m	output length
pwd	password	γ	public input
s	salt	H	crypto. hash function
g_{low}	min. garlic	$flap$	core of CATENA
g_{high}	garlic		

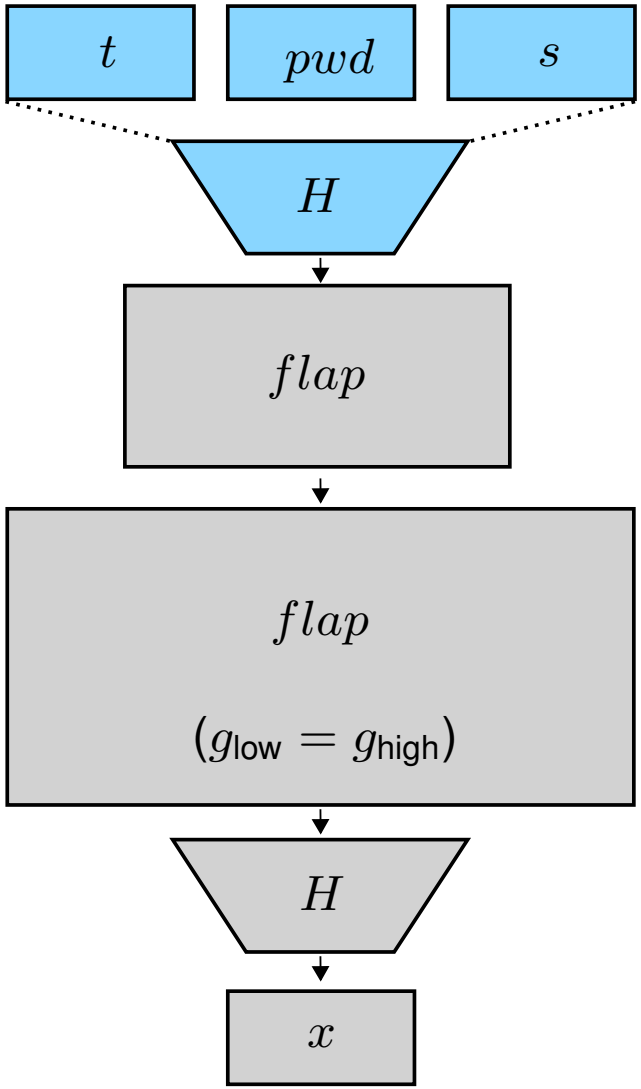


CATENA – The Framework

Algorithm 1 CATENA

1: $x \leftarrow H(t \mid\mid pwd \mid\mid s)$ ▷
2: $x \leftarrow flap(\lceil g_{\text{low}}/2 \rceil, x, \gamma)$
3: **for** $g = g_{\text{low}}, \ldots, g_{\text{high}}$ **do**
4: $\quad x \leftarrow flap(g, x \mid\mid 0^*, \gamma)$
5: $\quad x \leftarrow H(g \mid\mid x)$
6: $\quad x \leftarrow truncate(x, m)$
7: **end for**
8: **return** x

- make the password hash unique (tweak)
- 1-pass (pwd required only once)
- output of H is random value



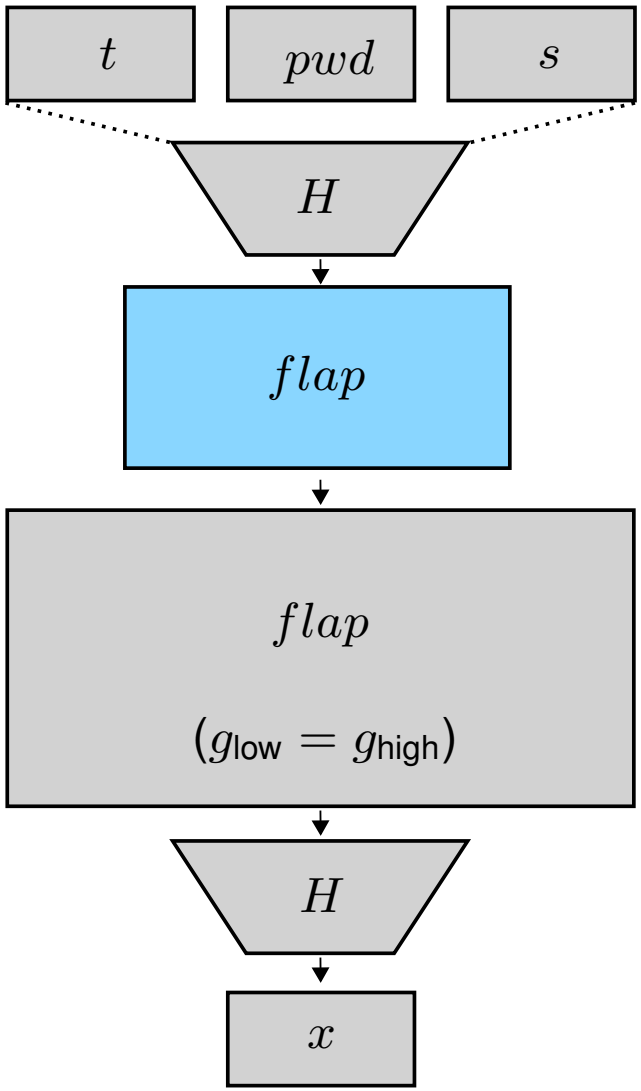
CATENA – The Framework

Algorithm 1 CATENA

1: $x \leftarrow H(t \mid\mid pwd \mid\mid s)$
2: $x \leftarrow flap(\lceil g_{\text{low}}/2 \rceil, x, \gamma)$ ▷
3: **for** $g = g_{\text{low}}, \ldots, g_{\text{high}}$ **do**
4: $\quad x \leftarrow flap(g, x \mid\mid 0^*, \gamma)$
5: $\quad x \leftarrow H(g \mid\mid x)$
6: $\quad x \leftarrow truncate(x, m)$
7: **end for**
8: **return** x

- overwrite the password-derived value significantly fast
- password itself has to be overwritten by implementation

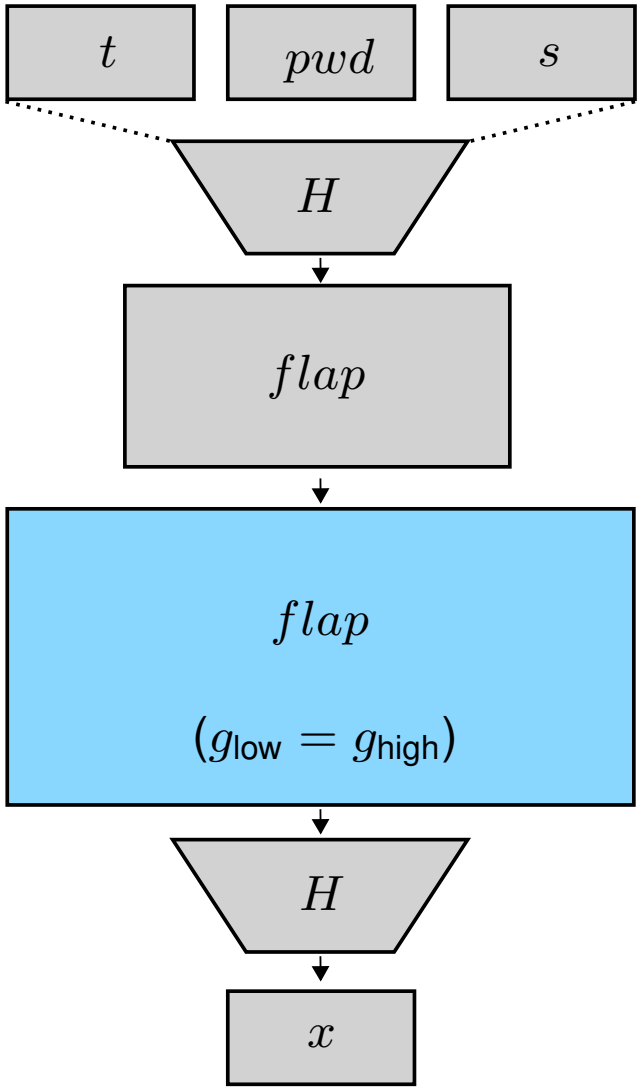
$\Rightarrow$ resistance against WGCA



CATENA – The Framework

Algorithm 1 CATENA

1: $x \leftarrow H(t \mid\mid pwd \mid\mid s)$
2: $x \leftarrow flap(\lceil g_{\text{low}}/2 \rceil, x, \gamma)$
3: **for** $g = g_{\text{low}}, \ldots, g_{\text{high}}$ **do**
4: $\quad x \leftarrow flap(g, x \mid\mid 0^*, \gamma)$ ▷
5: $\quad x \leftarrow H(g \mid\mid x)$
6: $\quad x \leftarrow truncate(x, m)$
7: **end for**
8: **return** x



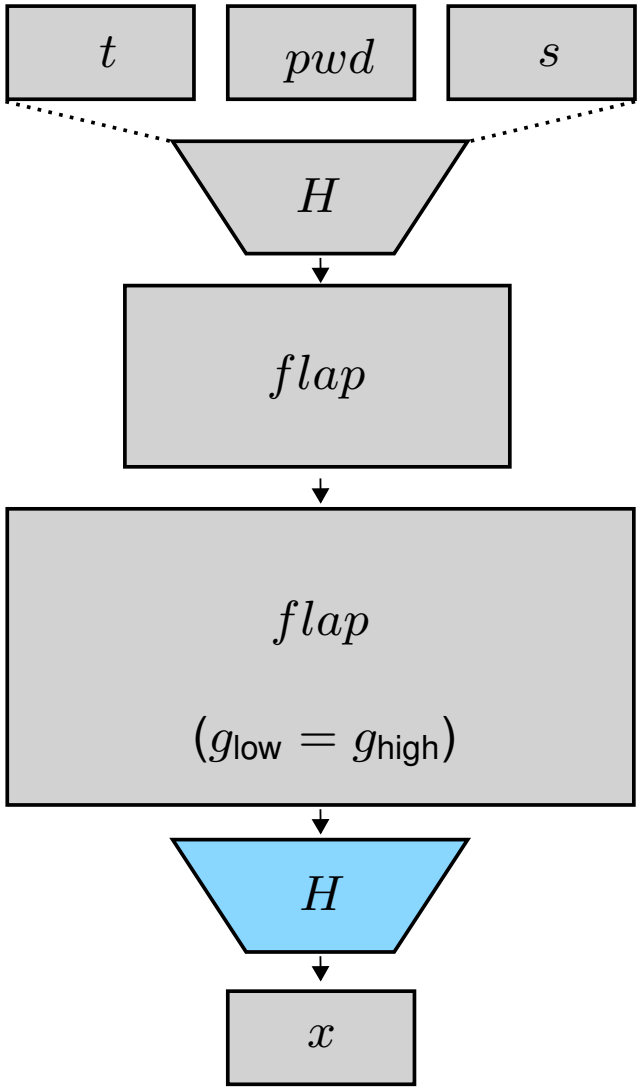
- core of CATENA (time and memory)
- graph-based structure
- memory-hardness
- highly flexible

CATENA – The Framework

Algorithm 1 CATENA

1: $x \leftarrow H(t \mid\mid pwd \mid\mid s)$
2: $x \leftarrow flap(\lceil g_{\text{low}}/2 \rceil, x, \gamma)$
3: **for** $g = g_{\text{low}}, \ldots, g_{\text{high}}$ **do**
4: $\quad x \leftarrow flap(g, x \mid\mid 0^*, \gamma)$
5: $\quad x \leftarrow H(g \mid\mid x)$ ▷
6: $\quad x \leftarrow truncate(x, m)$
7: **end for**
8: **return** x

- output of H is random
- preimage resistance
- server relief

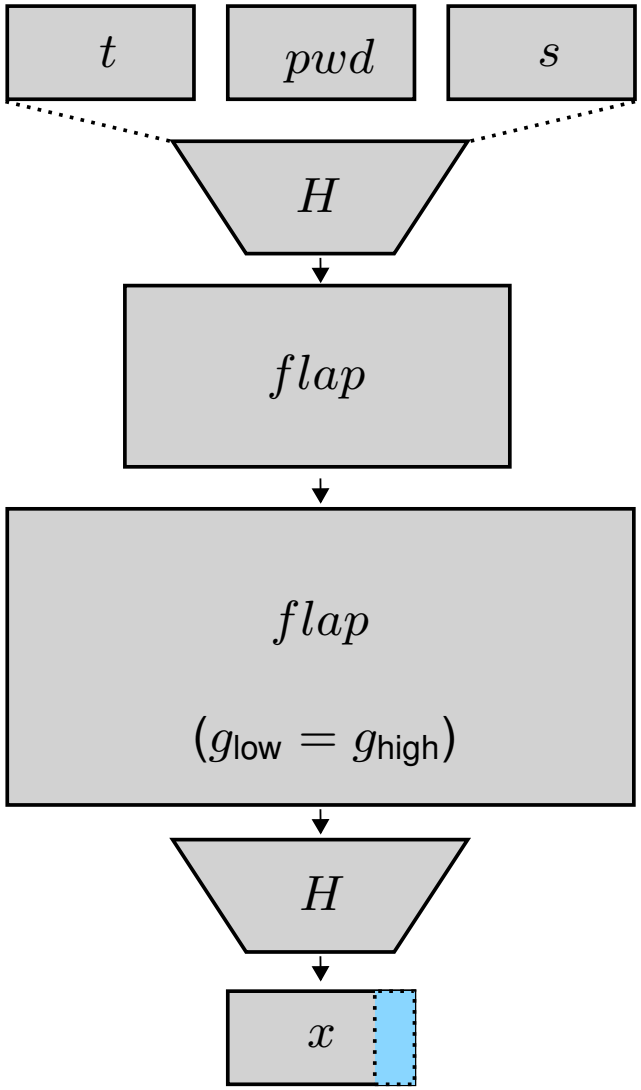


CATENA – The Framework

Algorithm 1 CATENA

1: $x \leftarrow H(t \mid\mid pwd \mid\mid s)$
2: $x \leftarrow flap(\lceil g_{\text{low}}/2 \rceil, x, \gamma)$
3: **for** $g = g_{\text{low}}, \ldots, g_{\text{high}}$ **do**
4: $\quad x \leftarrow flap(g, x \mid\mid 0^*, \gamma)$
5: $\quad x \leftarrow H(g \mid\mid x)$
6: $\quad x \leftarrow truncate(x, m)$ ▷
7: **end for**
8: **return** x

- can save space / memory
- possible threat (collisions)

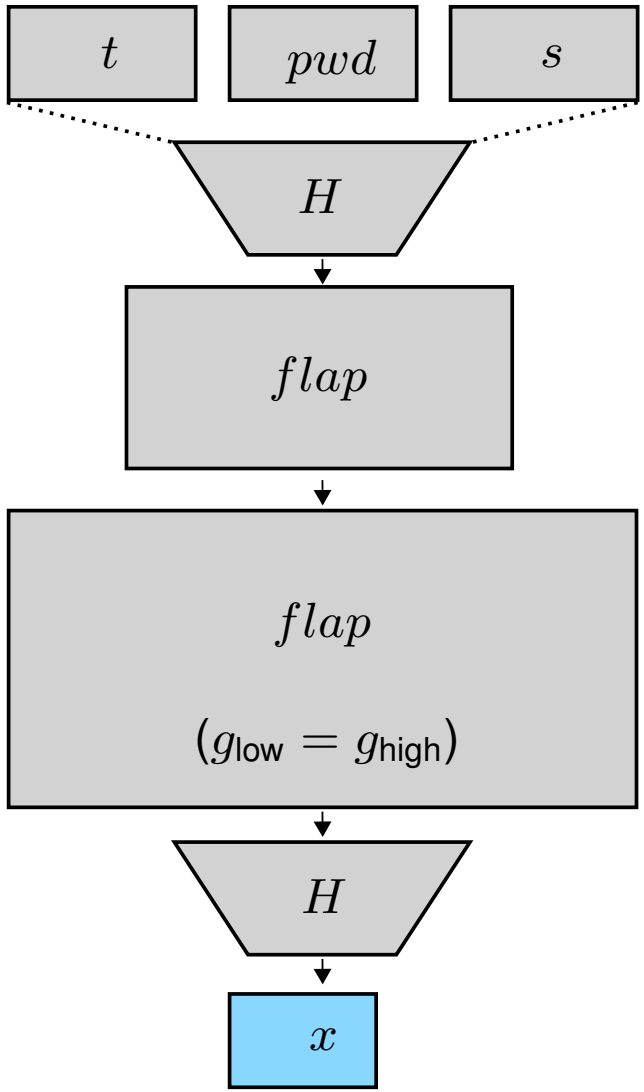


CATENA – The Framework

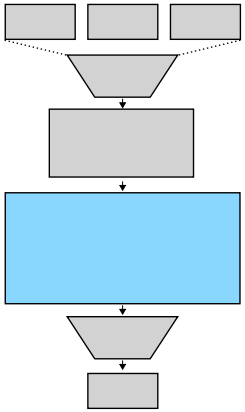
Algorithm 1 CATENA

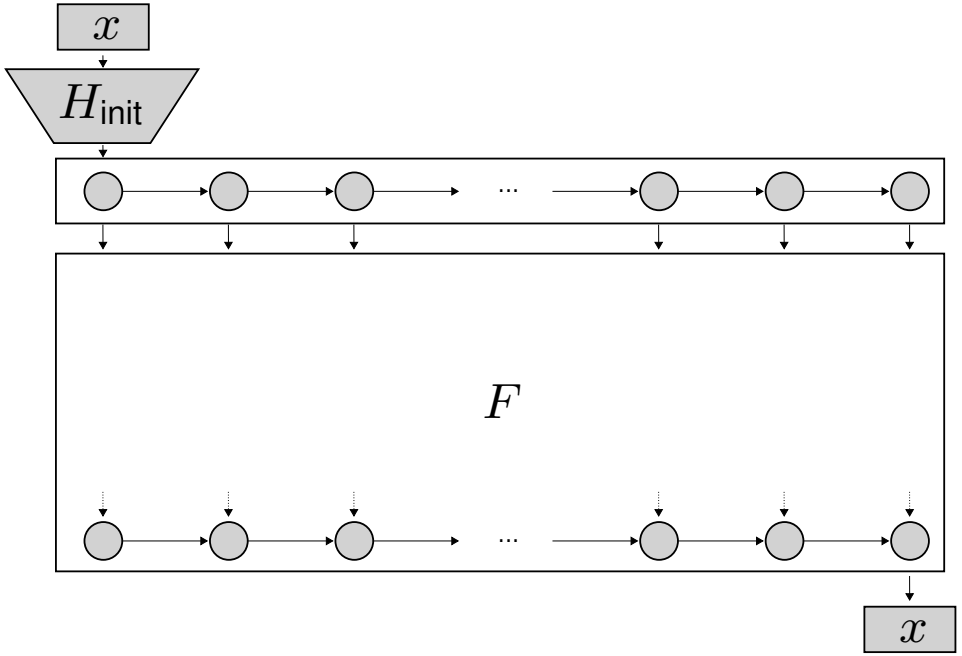
1: $x \leftarrow H(t \mid\mid pwd \mid\mid s)$
2: $x \leftarrow flap(\lceil g_{\text{low}}/2 \rceil, x, \gamma)$
3: **for** $g = g_{\text{low}}, \ldots, g_{\text{high}}$ **do**
4: $\quad x \leftarrow flap(g, x \mid\mid 0^*, \gamma)$
5: $\quad x \leftarrow H(g \mid\mid x)$
6: $\quad x \leftarrow truncate(x, m)$
7: **end for**
8: **return** x ▷

t	tweak	m	output length
pwd	password	γ	public input
s	salt	H	crypto. hash function
g_{low}	min. garlic	$flap$	core of CATENA
g_{high}	garlic		



CATENA – The Function $flap$

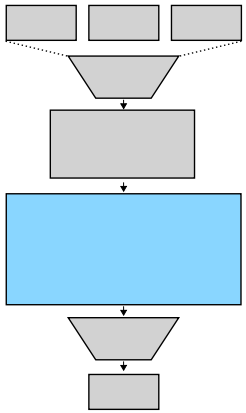


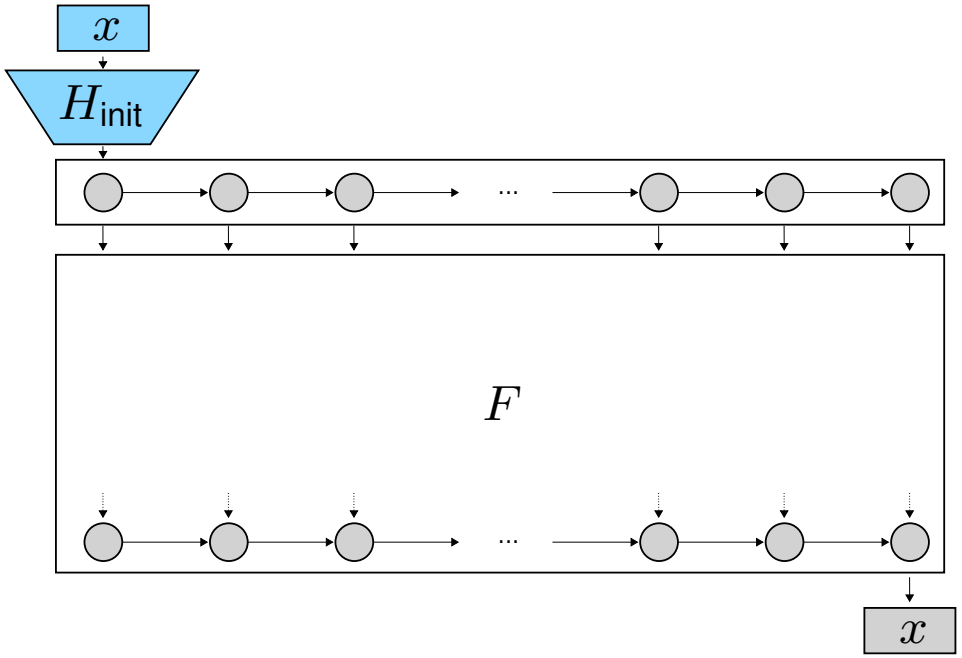


Algorithm 2 $flap$

1: $(v_{-2}, v_{-1}) \leftarrow H_{\text{init}}(x)$
2: **for** $i = 0, \ldots, 2^g - 1$ **do**
3: $\quad v_i \leftarrow H'(v_{i-1} \mid\mid v_{i-2})$
4: **end for**
5: $x \leftarrow F(v)$
6: **return** x

CATENA – The Function $flap$



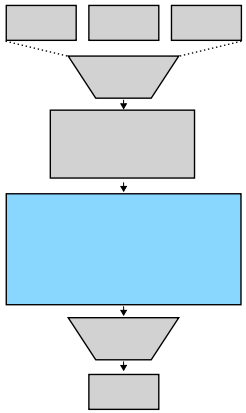


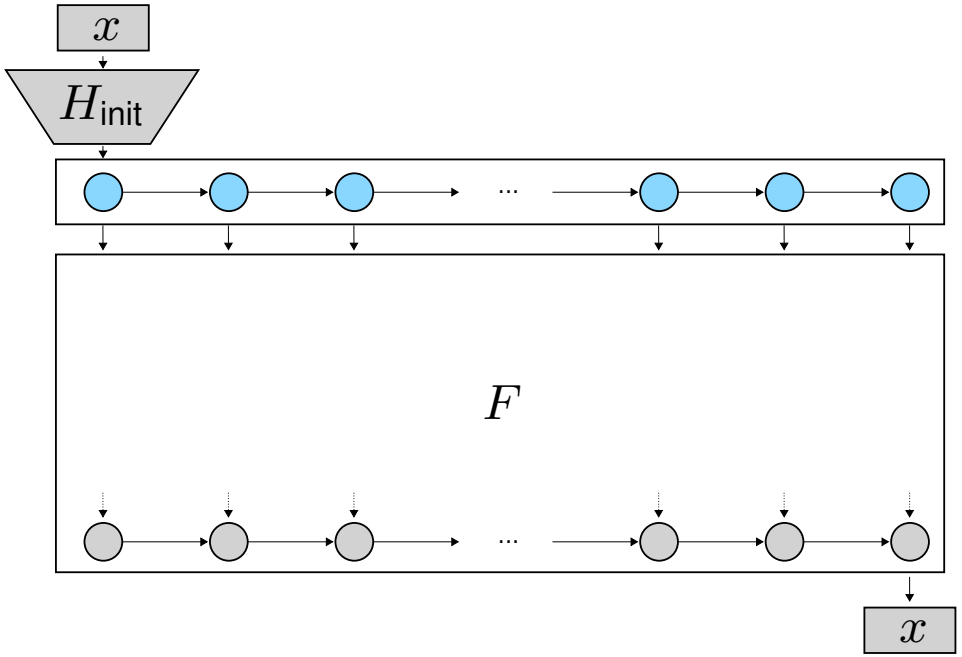
Algorithm 2 $flap$

1: $(v_{-2}, v_{-1}) \leftarrow H_{\text{init}}(x)$ ▷
2: **for** $i = 0, \ldots, 2^g - 1$ **do**
3: $\quad v_i \leftarrow H'(v_{i-1} \mid\mid v_{i-2})$
4: **end for**
5: $x \leftarrow F(v)$
6: **return** x

- H_{init} produces state words of arbitrary size
- becomes important if $|H(x)| \neq |H'(x)|$

CATENA – The Function $flap$



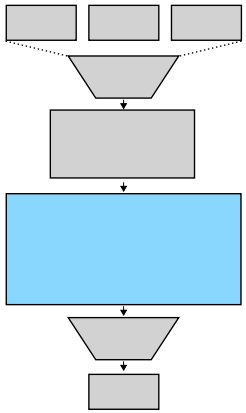


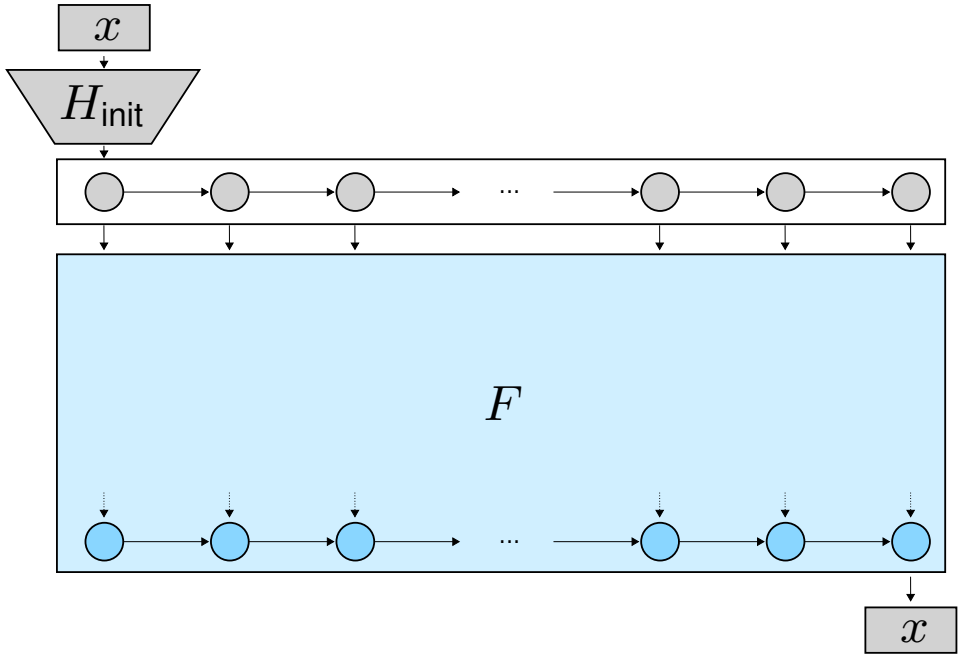
Algorithm 2 $flap$

1: $(v_{-2}, v_{-1}) \leftarrow H_{\text{init}}(x)$
2: **for** $i = 0, \ldots, 2^g - 1$ **do** ▷
3: $\quad v_i \leftarrow H'(v_{i-1} \mid\mid v_{i-2})$ ▷
4: **end for** ▷
5: $x \leftarrow F(v)$
6: **return** x

- sequential initialization of 2^g n-bit words
- usage of H' to reduce computational effort

CATENA – The Function $flap$



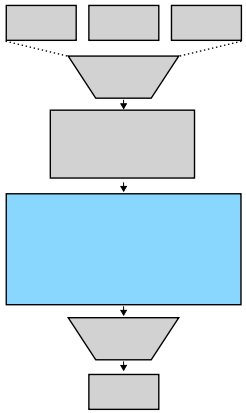


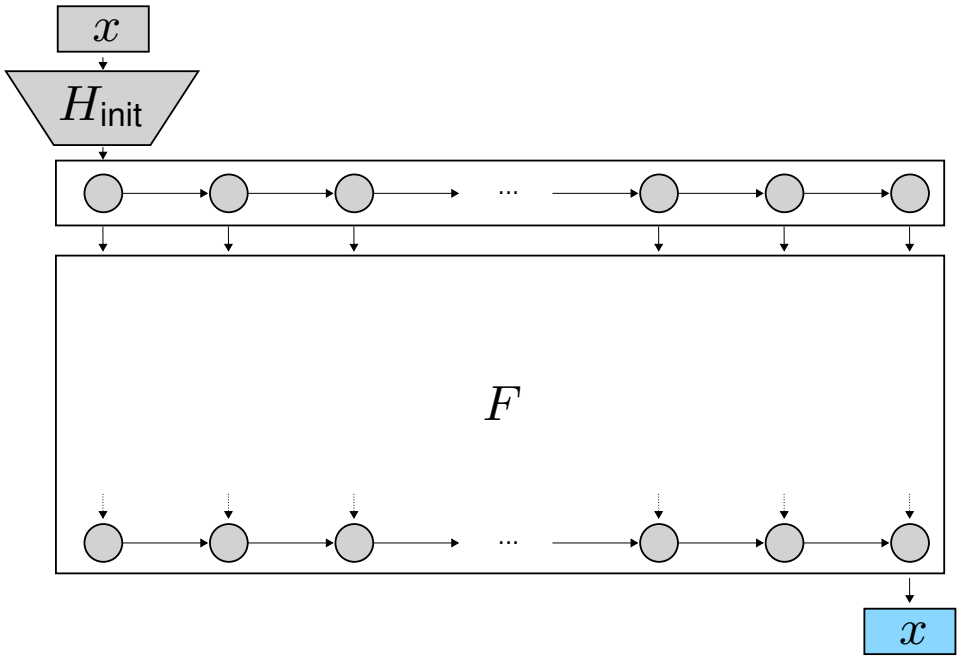
Algorithm 2 $flap$

1: $(v_{-2}, v_{-1}) \leftarrow H_{\text{init}}(x)$
2: **for** $i = 0, \ldots, 2^g - 1$ **do**
3: $\quad v_i \leftarrow H'(v_{i-1} \mid\mid v_{i-2})$
4: **end for**
5: $x \leftarrow F(v)$ ▷
6: **return** x

- graph-based memory-hard core of CATENA
- password-independent indexing
- replaced by certain graph instance
- usage of H' to reduce computational effort

CATENA – The Function $flap$





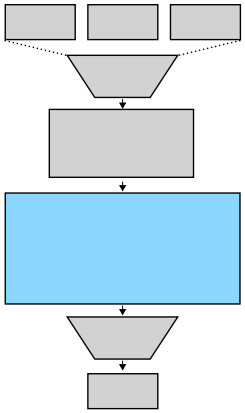
Algorithm 2 $flap$

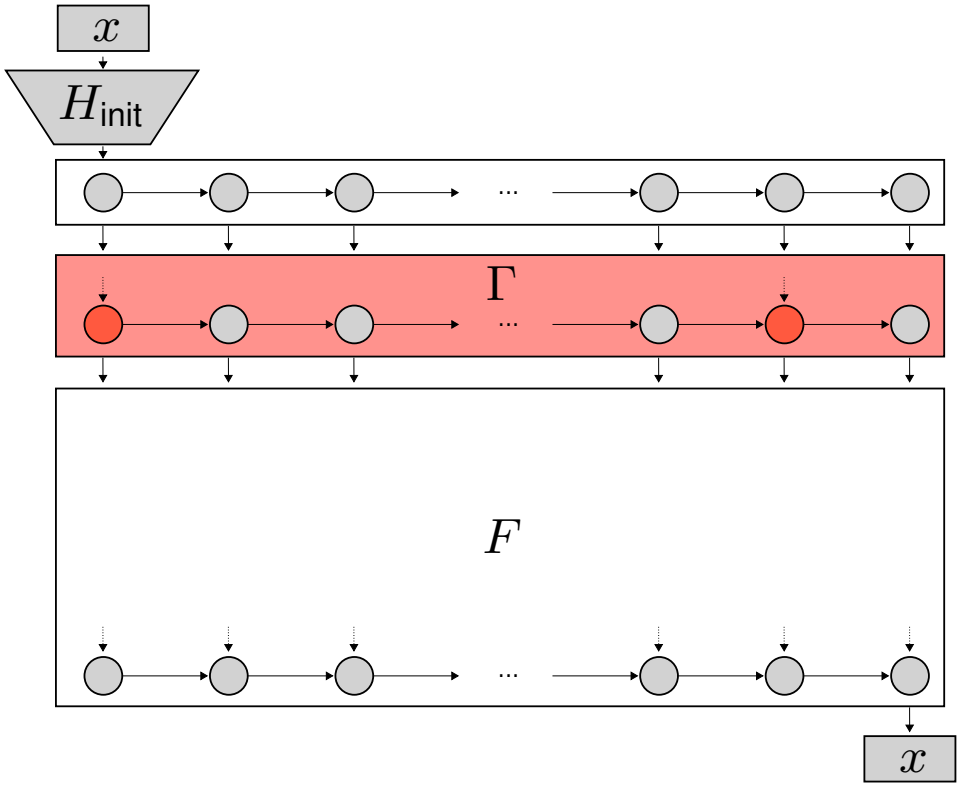
1: $(v_{-2}, v_{-1}) \leftarrow H_{\text{init}}(x)$
2: **for** $i = 0, \ldots, 2^g - 1$ **do**
3: $\quad v_i \leftarrow H'(v_{i-1} \mid\mid v_{i-2})$
4: **end for**
5: $x \leftarrow F(v)$
6: **return** x ▷

Section 3

Extensions to CATENA

CATENA – Extensions to $flap$



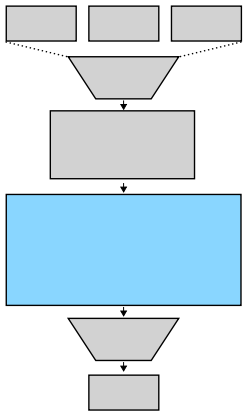


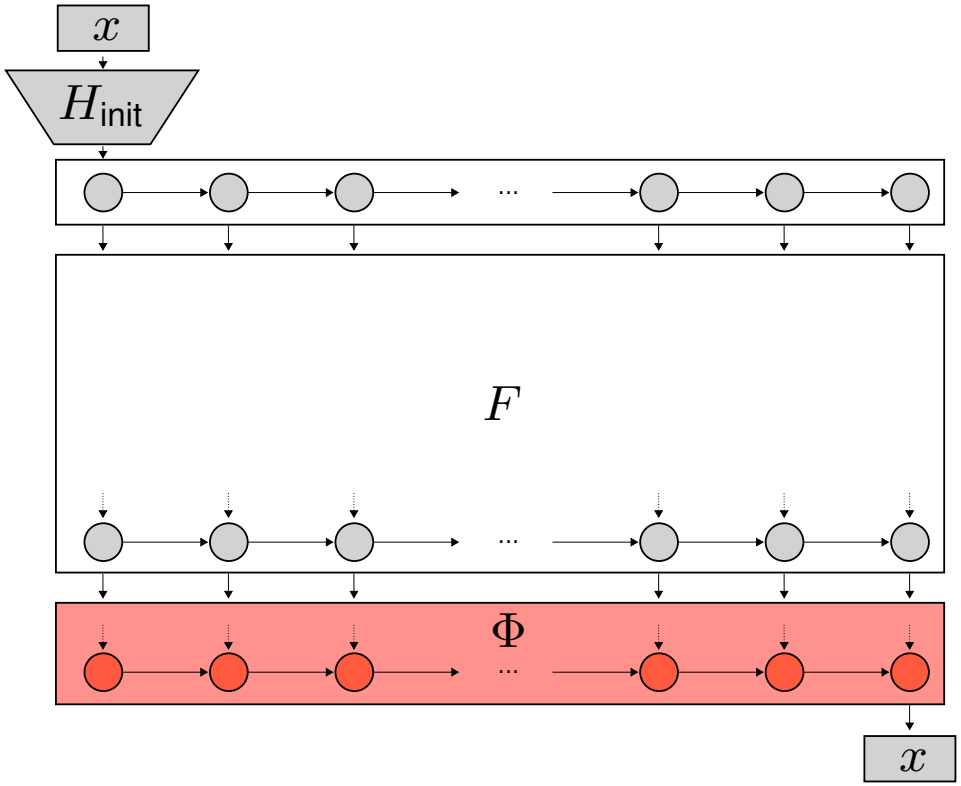
Algorithm 3 Extension Γ

1: $r \leftarrow H(\gamma) \mid\mid H(H(\gamma))$
2: $p \leftarrow 0$
3: **for** $i \leftarrow 0, \ldots, 2^{\lceil 3g/4 \rceil} - 1$ **do**
4: $\quad (j_1, r, p) \leftarrow R(r, p)$
5: $\quad (j_2, r, p) \leftarrow R(r, p)$
6: $\quad v_{j_1} \leftarrow H'(v_{j_1} \mid\mid v_{j_2})$
7: **end for**
8: **return** v

- overwrites $2^{\lceil 3g/4 \rceil}$ randomly chosen state words
- indexing is salt-dependent (or any **public** input)
- resistance against ASICs

CATENA – Extensions to $flap$





Algorithm 4 Extension Φ

1: $j \leftarrow R(\mu)$
2: $v_0 \leftarrow H'(v_{2^g-1} \mid\mid v_j)$
3: **for** $i \leftarrow 1, \ldots, 2^g - 1$ **do**
4: $\quad j \leftarrow R(v_{i-1})$
5: $\quad v_i \leftarrow H'(v_{i-1} \mid\mid v_j)$
6: **end for**
7: **return** v

- indexing is **secret**-dependent (`scrypt`-like)
- resistance against ASICs/GPUs
- resistance against tradeoff attacks
- allows for cache-timing attacks (not efficiently)

Section 4

Graph Structures

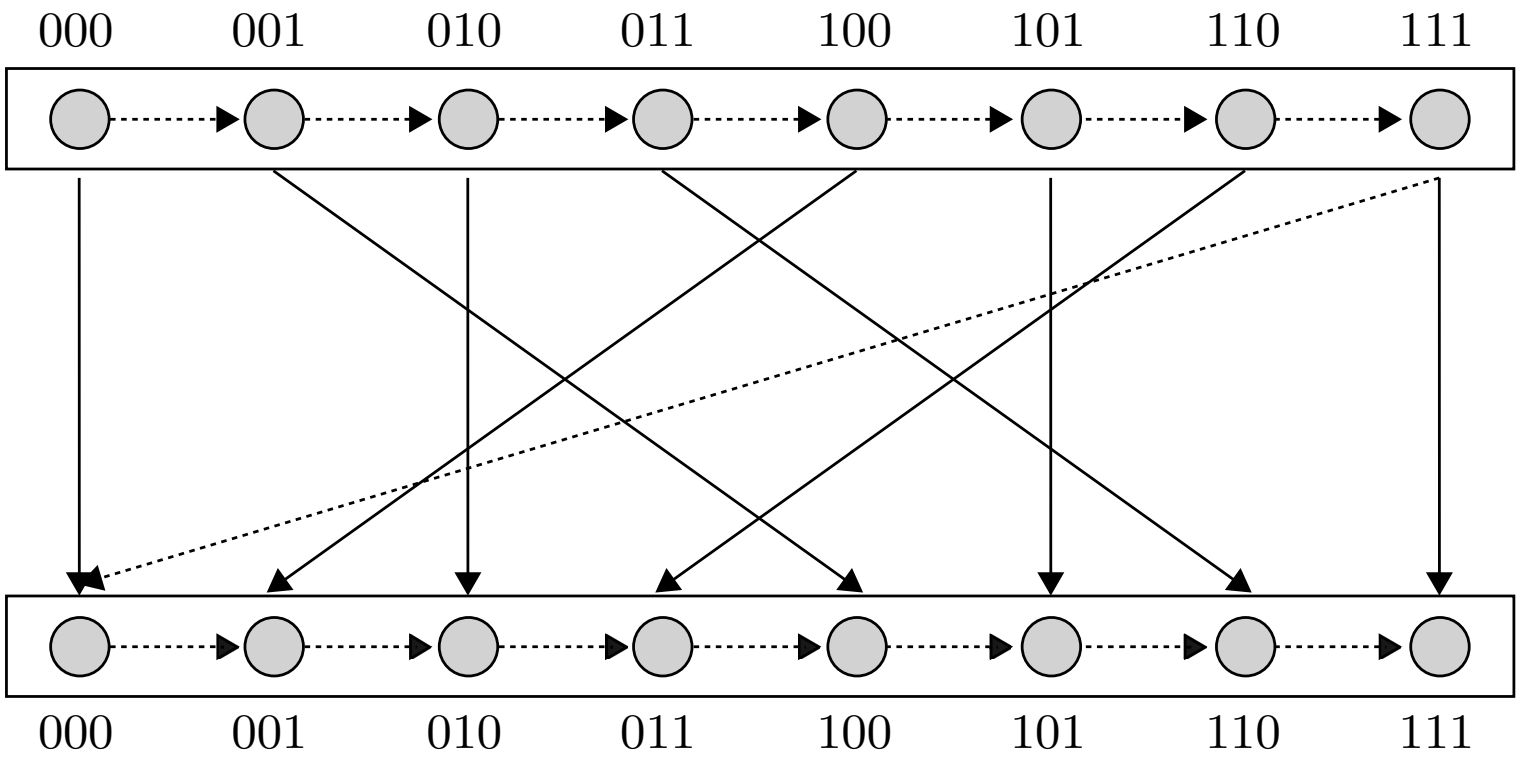
Bit-Reversal Graph (BRG)

Idea (borrowed from Lengauer and Tarjan)

(Thomas Lengauer and Robert Ende Tarjan. Asymptotically Tight Bounds on Time-Space Trade-offs in a Pebble Game. *J. ACM*, 1982.)

Construct a (hash) graph that exploits the bit-reversal permutation τ:

$$\tau(i_0, i_1, \ldots, i_{g-1}) = (i_{g-1}, i_{g-2}, \ldots, i_0)$$

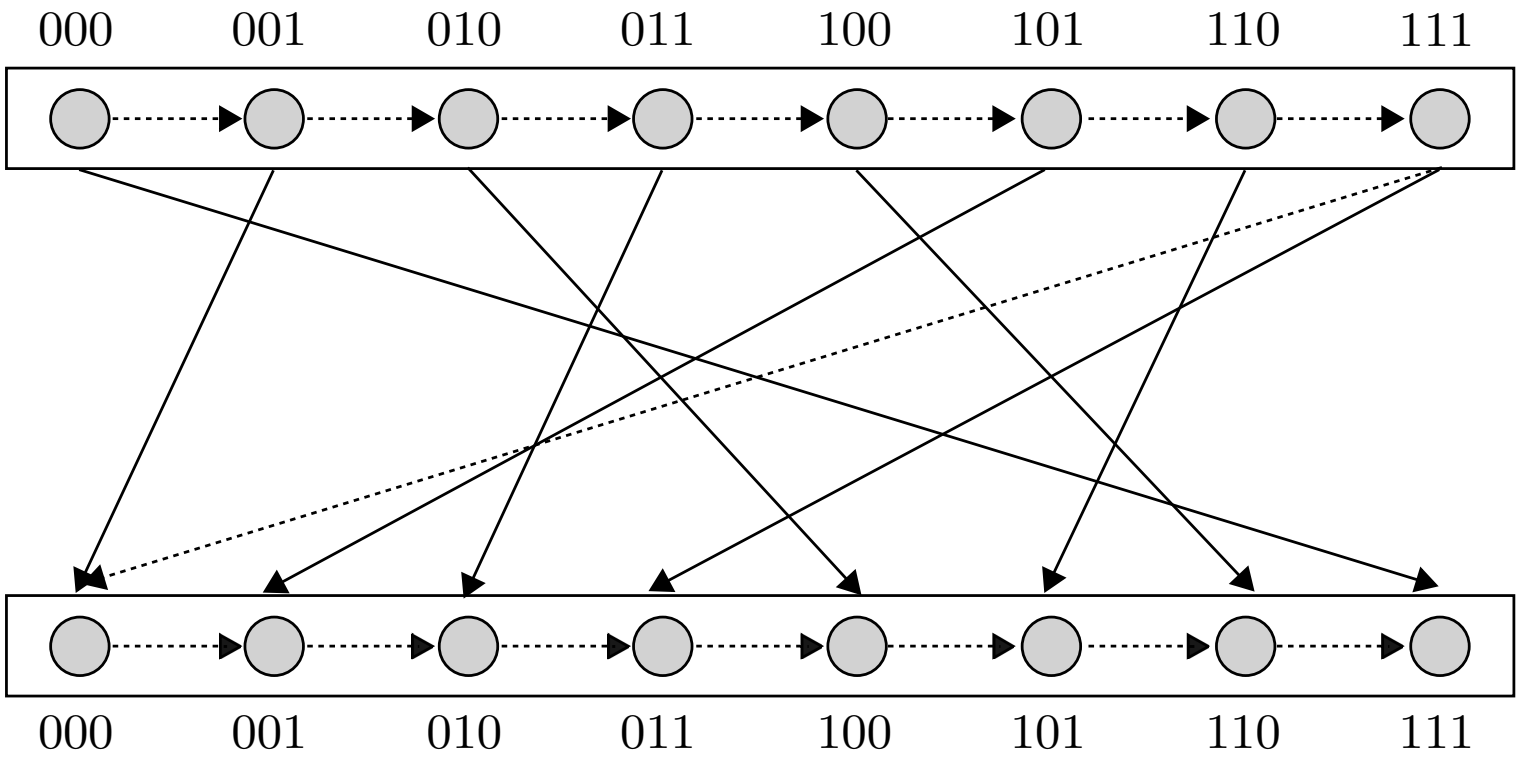


Shifted Bit-Reversal Graph (SBRG)

Shifted Bit-Reversal Graph (SBRG)

Construct a (hash) graph that exploits the shifted bit-reversal permutation τ_s:

$$\tau_s(i) = (\tau(i) + c) \bmod 2^g$$

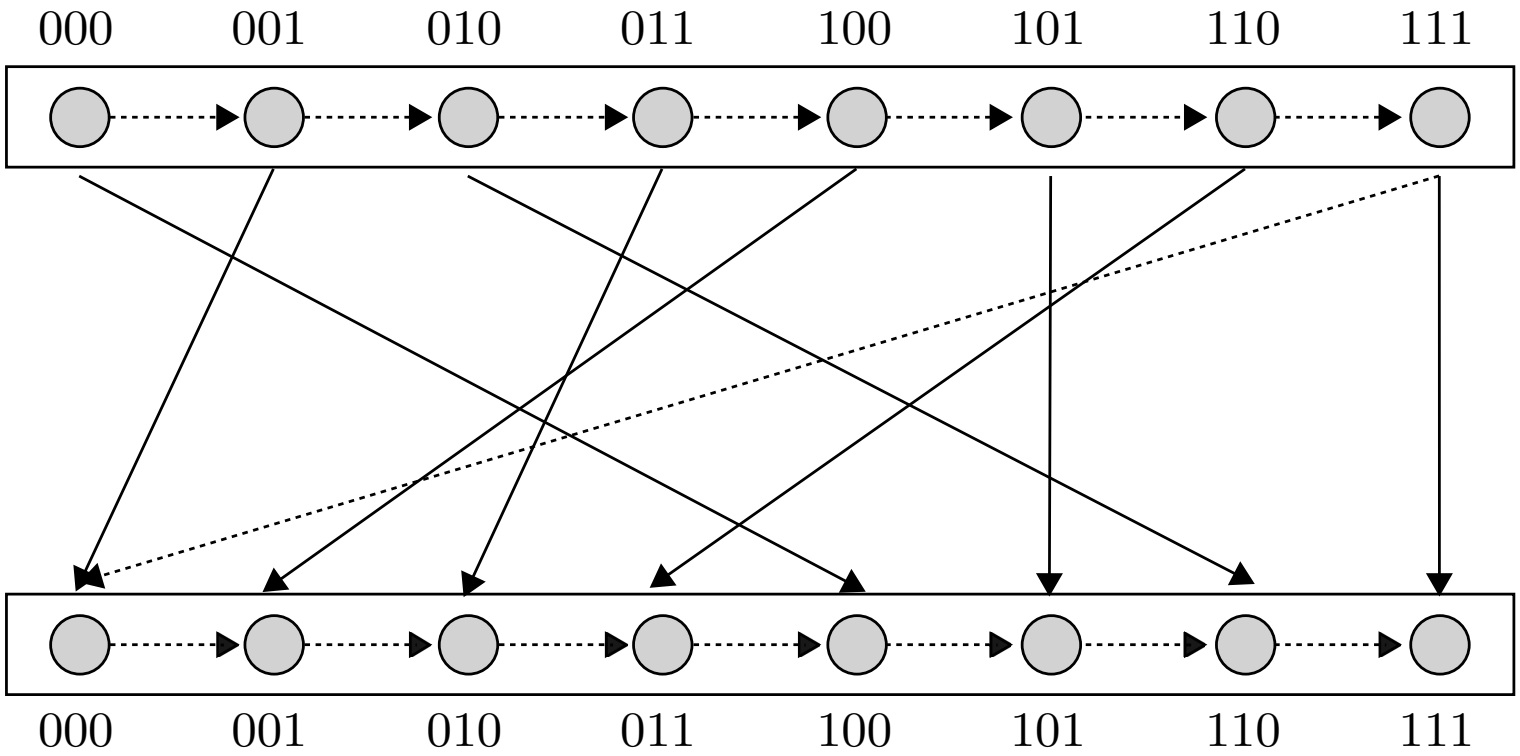


Gray-Reverse Graph (GR)

Gray Reverse (GR2/GR3)

Construct a (hash) graph that exploits the following indexing function τ_x:

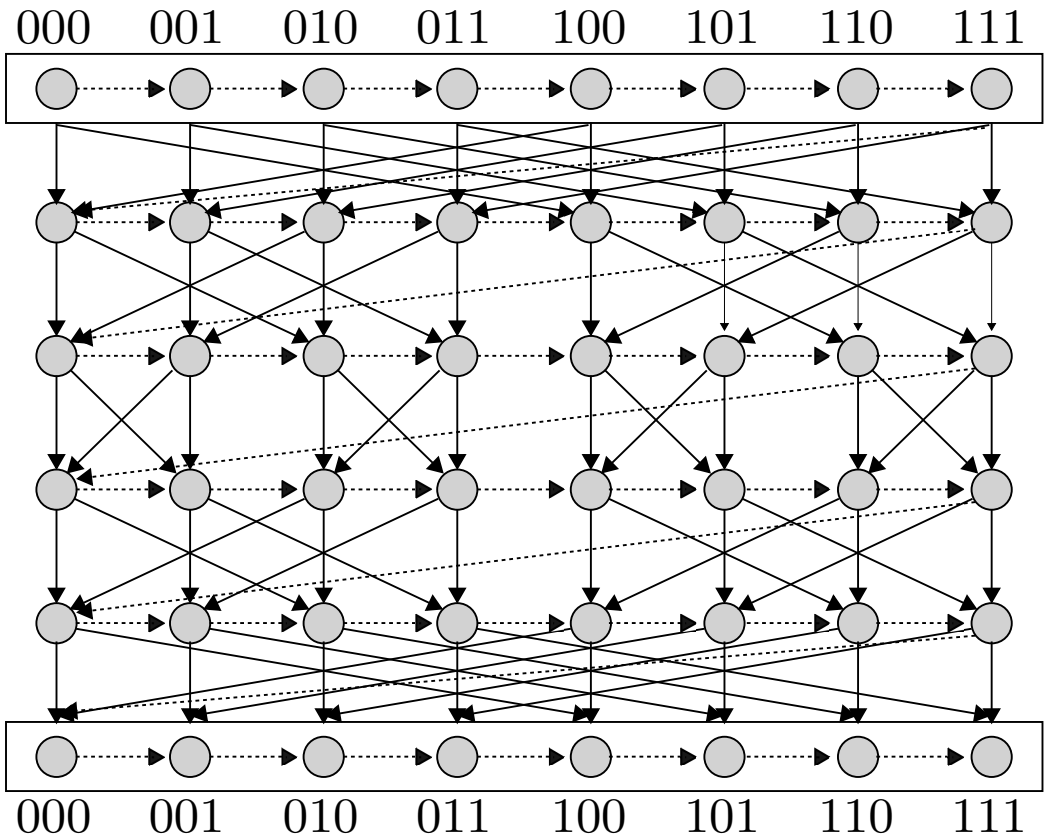
$$\tau_x(i) = \tau(i) \oplus \left(\overline{\tau(i)} \gg \left\lceil \frac{g}{x} \right\rceil\right)$$



Double-Butterfly Graph (DBG)

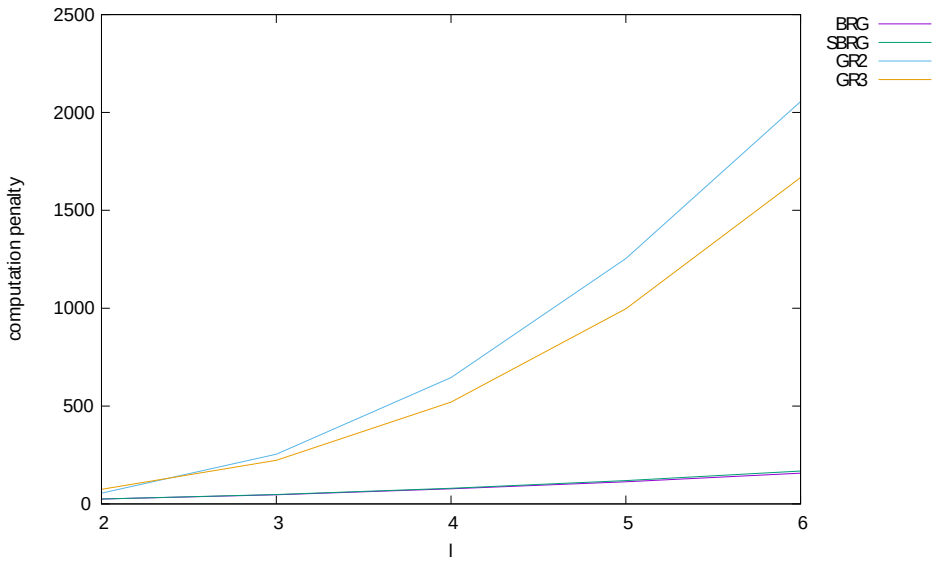
Double Butterfly Graph (DBG)

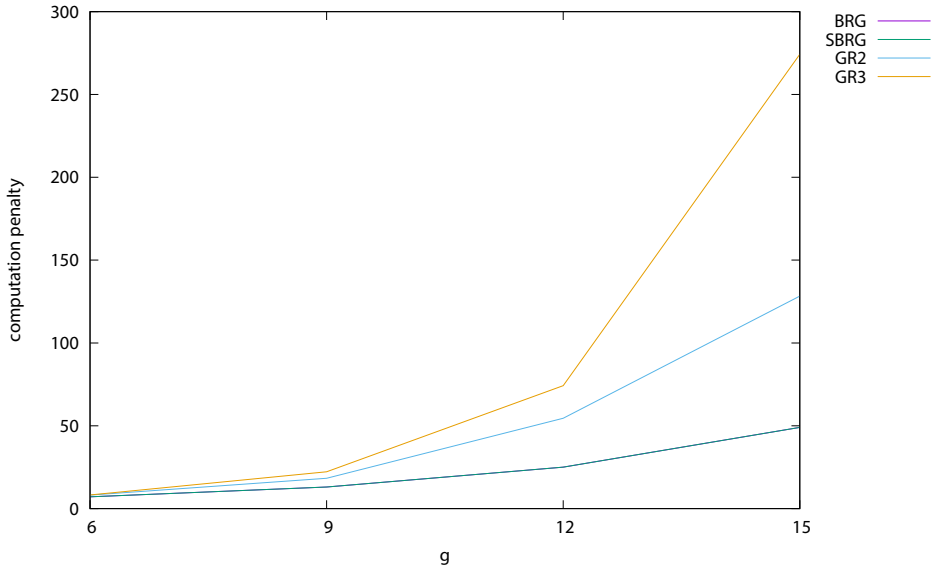
Two interlocked Cooley-Tukey FFT Graphs (omitting one middle row)



Comparison of Graph Instantiations

- Penalty of an adversary regarding the precomputation attacks[1]
- every fourth state value (of each row) is a *precomputed* sampling point



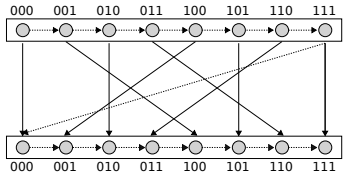


- fix $g = 12$
- variable layer amount (λ)
- using extension Γ

- fix $\lambda = 2$
- variable garlic (g)
- using extension Γ

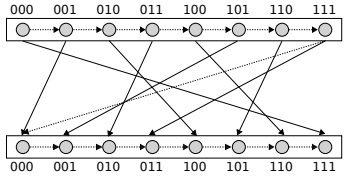
[1] Alex Biryukov and Dmitry Khovratovich. Tradeoff Cryptanalysis of Memory-Hard Functions, *IACR Cryptology ePrint Archive,* 2015

Comparison of Graph Instantiations



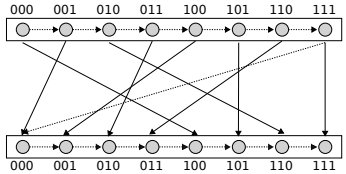
Bit-Reversal Graph

- 1-mem.-hard, fast
- low penalty for tradeoff adversary



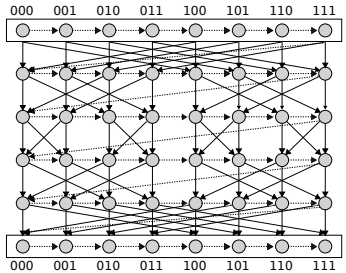
Shifted BRG

- 1-mem.-hard, tunable
- low penalty for tradeoff adversary



Gray-Reverse Graph

- 1-mem.-hard
- reasonable penalty for tradeoff adversary



Double-Butterfly Graph

- λ-mem.-hard, slow
- high penalty for tradeoff adversary

Section 5

Hash Functions

Hash-Function Instances

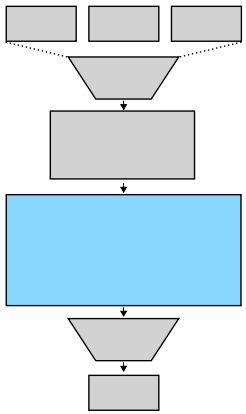
- H is always a cryptographic hash function, e.g., BLAKE2b
- possibilities for H':
  - equal to H or any other cryptographic hash function (required for random-oracle security – key derivation)
  - reduced hash function
- table: $\lambda = 2$, 128 MB of memory, no extensions

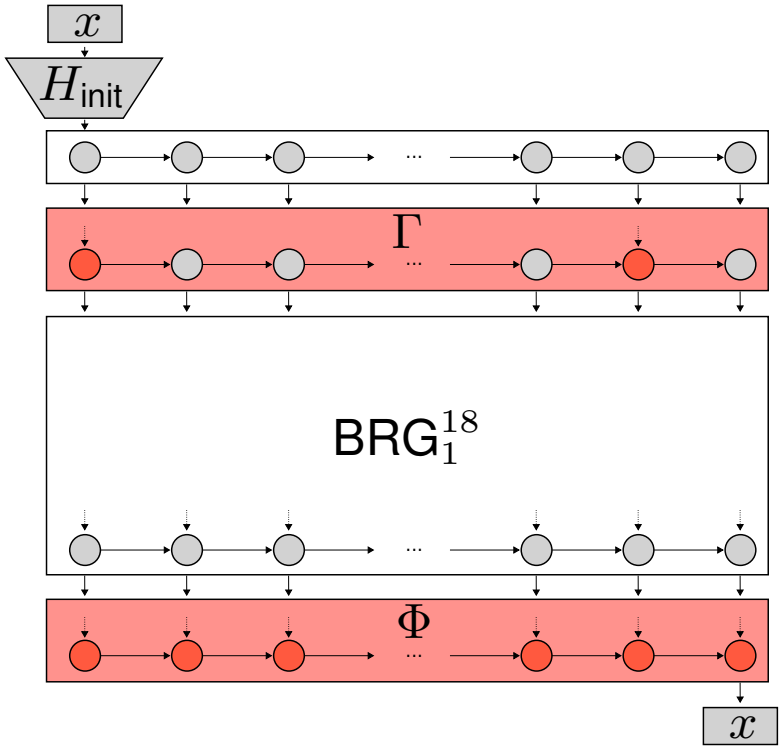
Fast HF (H')	**Garlic** (g)	**Time (s)**	**Throughput (GB/s)**
BLAKE2b	21	1.55	0.08
BLAKE2b-1 (G_O)	21	0.37	0.34
CF (G_B)	17	0.25	0.50
CF (G_L)	17	0.18	0.69
GF	21	0.64	0.19
MultHash	21	0.30	0.42
SHA-512	21	4.04	0.03

Section 6

New Recommendations

CATENA-STONEFLY



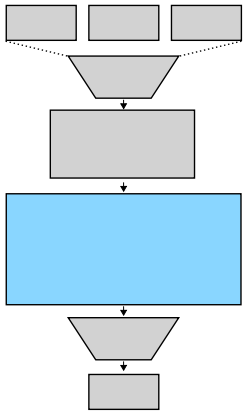


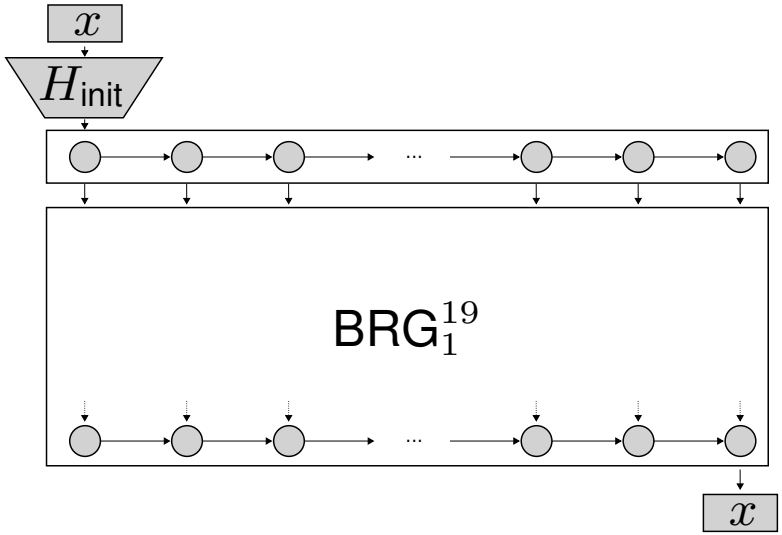
Algorithm 5

1: $(v_{-2}, v_{-1}) \leftarrow H_{\text{init}}(x)$
2: **for** $i = 0, \ldots, 2^g - 1$ **do**
3: $\quad v_i \leftarrow H'(v_{i-1} \mid\mid v_{i-2})$
4: **end for**
5: $v \leftarrow \Gamma(g, v, \gamma)$
6: $v \leftarrow \text{BRG}_1^{18}(v)$
7: $x \leftarrow \Phi(g, v, \mu)$
8: **return** x

- resistance against ASICs/GPUs (Γ and Φ)
- sequential memory-hard (Φ)
- multiplication hardening – CF (G_B)
- allows for cache-timing attacks (Φ)

CATENA-HORSEFLY



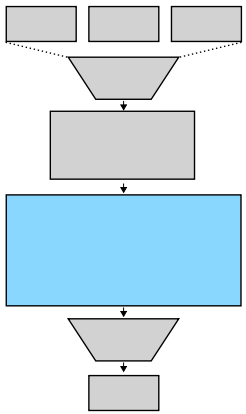


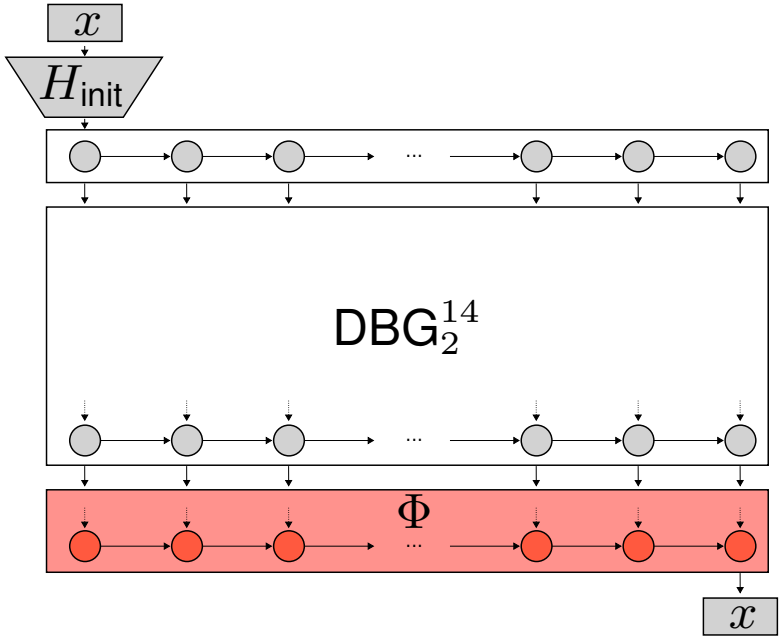
Algorithm 6

1: $(v_{-2}, v_{-1}) \leftarrow H_{\text{init}}(x)$
2: **for** $i = 0, \ldots, 2^g - 1$ **do**
3: $\quad v_i \leftarrow H'(v_{i-1} \mid\mid v_{i-2})$
4: **end for**
5: $v \leftarrow \text{BRG}_1^{19}(v)$
6: **return** x

- fastest instance (throughput $\sim 1\,\text{GB/s}$)
- allows for GC attacks (set $\lambda = 2$ to avoid these)
- deploys CF (G_L)

CATENA-MYDASFLY



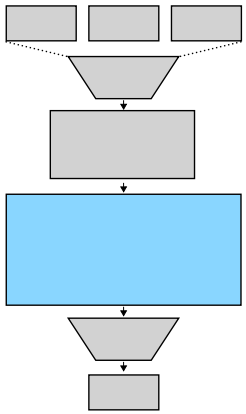


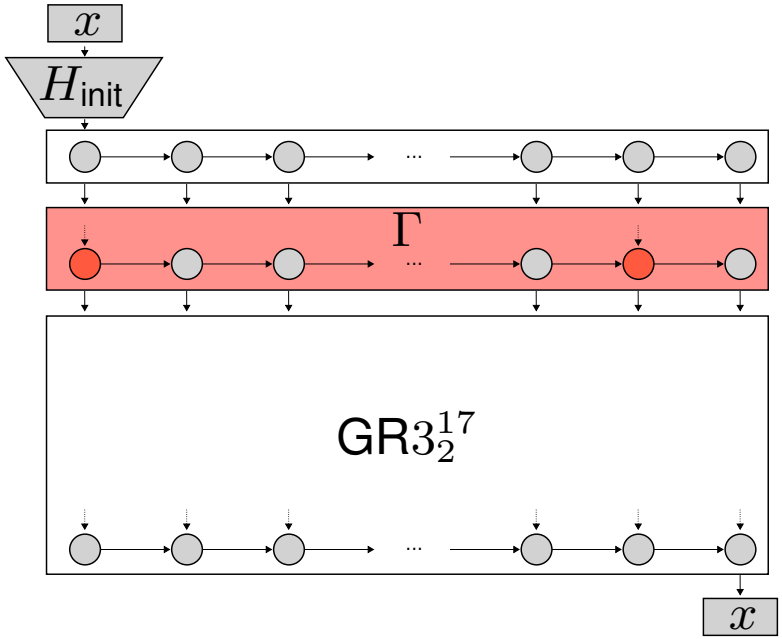
Algorithm 7

1: $(v_{-2}, v_{-1}) \leftarrow H_{\text{init}}(x)$
2: **for** $i = 0, \ldots, 2^g - 1$ **do**
3: $\quad v_i \leftarrow H'(v_{i-1} \mid\mid v_{i-2})$
4: **end for**
5: $v \leftarrow \text{DBG}_2^{14}(v)$
6: $x \leftarrow \Phi(g, v, \mu)$
7: **return** x

- 2-memory-hardness through DBG_2^{14}
- increased to sequential memory-hardness by Φ
- allows for cache-timing attacks (Φ)
- deploys CF (G_L)

CATENA-LANTERNFLY



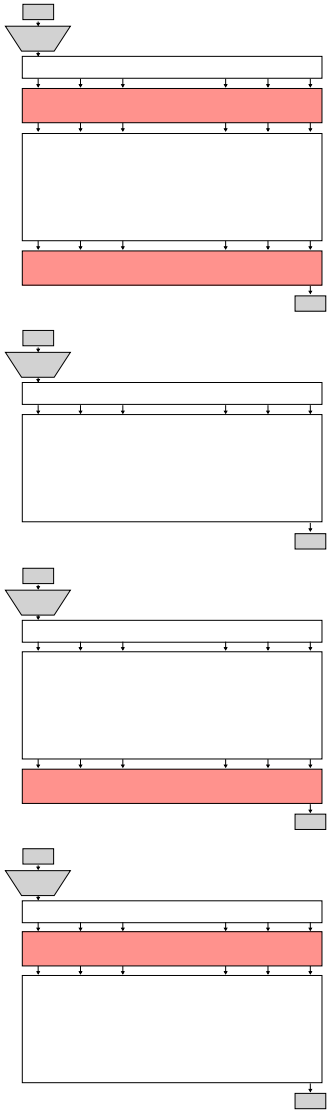


Algorithm 8

1: $(v_{-2}, v_{-1}) \leftarrow H_{\text{init}}(x)$
2: **for** $i = 0, \ldots, 2^g - 1$ **do**
3: $\quad v_i \leftarrow H'(v_{i-1} \mid\mid v_{i-2})$
4: **end for**
5: $v \leftarrow \Gamma(g, v, \gamma)$
6: $v \leftarrow F(v)$
7: **return** x

- resistance against ASICs (Γ)
- good resistance against tradeoff attacks (GR3_2^{22})
- multiplication-hardening – CF (G_B)

Comparison of Instantiations



CATENA-STONEFLY
(ASIC Resistant)

- seq. memory-hardness
- multiplication-hardening

CATENA-HORSEFLY
(High Throughput)

- fast graph traversing
- may allow GC attacks

CATENA-MYDASFLY
(Tradeoff Resistant)

- strong graph
- seq. memory-hardness

CATENA-LANTERNFLY
(Hybrid Approach)

- reasonable performance
- reasonable resistance

Section 7

Conclusion

Conclusion

- showed the high flexibility of CATENA
  - instantiations of H'
  - instantiations of F (graph-based core of CATENA)
  - extensions (Γ and Φ)

- extended the CATENA portfolio by four new instances for password hashing / key derivation[2]

- future work: find λ-memory-hard graph instance with constant depth

[2] For key derivation: proof for random-oracle security exists only *iff* $H = H'$

Questions?