Rotational cryptanalysis of round-reduced Keccak

Paweł Morawiecki [1,2] Josef Pieprzyk [3] Marian Srebrny [1,2]

[1]Section of Informatics, Kielce University of Commerce, Poland

[2]Institute of Computer Science, Polish Academy of Sciences, Poland

[3]Department of Computing, Macquarie University, Sydney, Australia

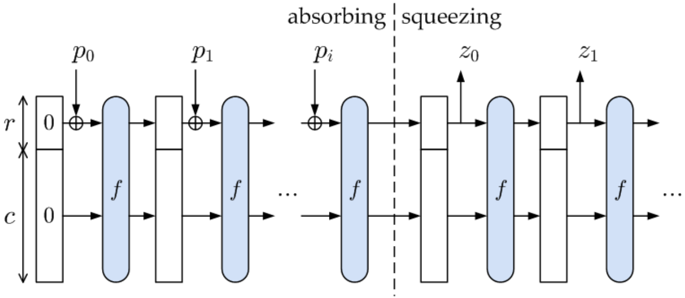
FSE'13, Singapore, 12.03.2013

Outline

1. Brief description of Keccak
2. Rotational cryptanalysis — main idea
3. Rotational analysis of Keccak-f[1600] permutation
4. Preimage attacks on 3- and 4-round Keccak
5. Conclusion

Outline

Keccak sponge function family



- Two main parameters: r (bitrate), c (capacity)
- For SHA-3 candidates, $r + c = 1600$ bits
- A security/performance trade-off by choosing r and c values

Keccak-f[1600] permutation

- Keccak-f[1600] permutation consists of 24 rounds.
- Each round has 5 steps: θ, ρ, π, χ, and ι.
- Rounds differ only in ι (different values of round constants).

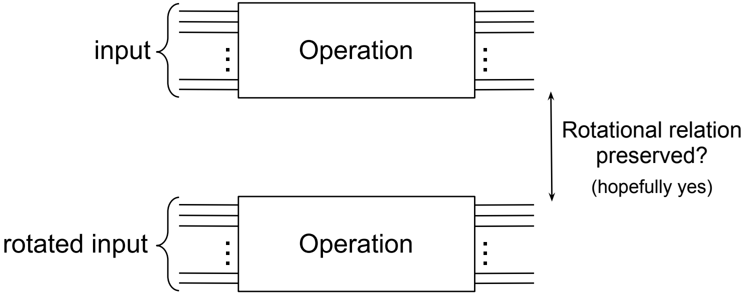
Steps of Keccak-f[1600]

- θ : a linear map, which adds to each bit in a column the parity of two other columns (only XORs)
- ρ : rotations inside 64-bit words (called 'lanes')
- π : permutation between whole lanes
- χ : the only non-linear mapping of Keccak, working on each of the 320 rows independently (ANDs and XORs)
- ι : one lane is XORed with a 64-bit constant (each round has a different constant)

Outline

Rotational cryptanalysis — main idea

- Bitwise rotation (8-bit word '00010100' rotated by 3 gives '10000010')
- In the rotational analysis, the adversary investigates the propagation of the rotational relations through the cryptographic primitive.



Bitwise XOR operation

8-bit inputs	inputs rotated by 2
00101000	00001010
XOR 00101011	XOR 11001010
00000011	11000000

- Bitwise XOR operation preserves rotational relation.

Bitwise AND operation

Two 8-bit inputs		inputs rotated by 2	
	00101000		00001010
AND	00101011	AND	11001010
	00101000		00001010

- Bitwise AND operation preserves rotational relation.

Bitwise rotation (circular shift) operation

8-bit input	input rotated by 2
00101000	00001010
rotate by 7	rotate by 7
01010000	00010100

- Bitwise rotation operation in a natural way preserves rotational relation.

XORing with constant — root of all evil

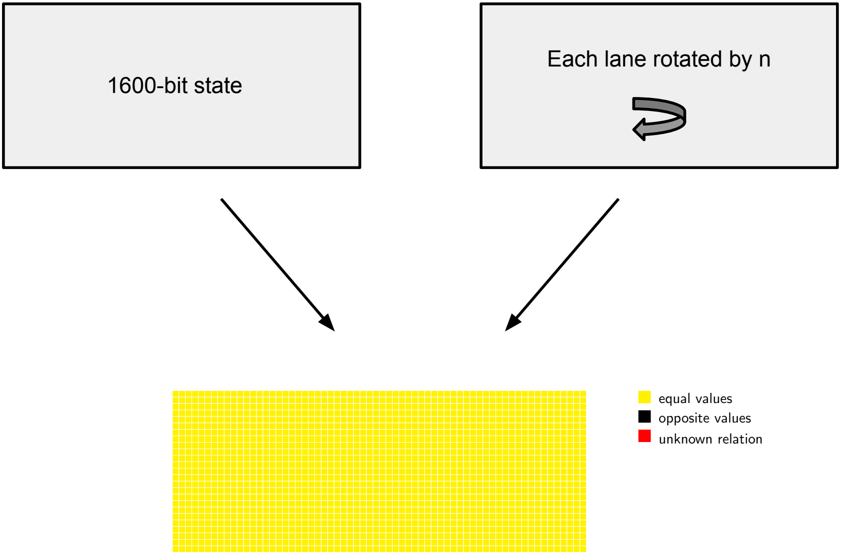
8-bit input		input rotated by 2
00101000		00001010
XOR *00000001*		XOR *00000001*
00101001		00001011

'inversions'

- XORing with constant does not preserve rotational relations.
- The more 1's in a constant, the more 'inversions' introduced.
- Luckily, in Keccak Hamming weights of constants are very low.

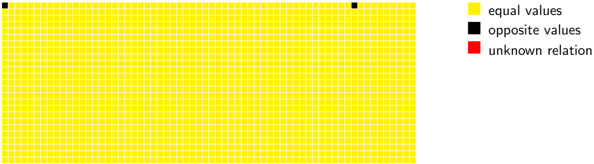
Outline

Rotational pair of states

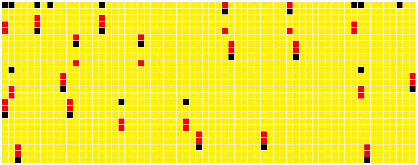


Evolution of a rotational pair

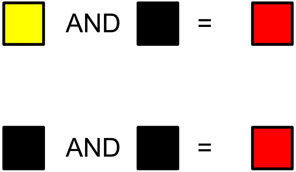
Round 1



Round 2



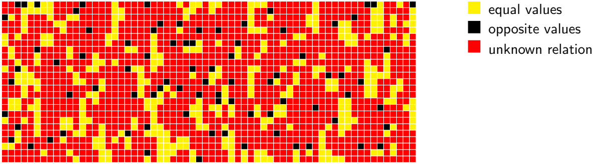
Non-linear χ and rotational relations



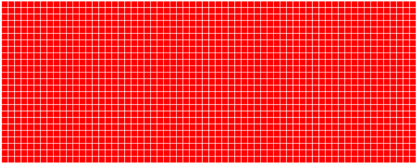
- Non-linear step χ introduces an uncertainty in rotational relation between the corresponding bits. (similar to differential cryptanalysis)

Evolution of a rotational pair

Round 3



Round 4



Outline

3-round preimage attack on Keccak-512

- Keccak-512 ($r = 576$, $c = 1024$, $hash = 512$)
- Message structure: first 8 lanes (512 bits) are unknown (to be determined by the attacker). Last 62 bits of the message are set to 1. The message is padded with two 1's giving a block of 576 bits.
  - It is expected that among 2^{512} possible messages there is, on average, one preimage of a given hash.
  - With such structure, to guess a rotational counterpart of a state we need to care only about 512 unknown bits. (A lane with all 0's or all 1's stays the same before and after rotation.)

3-round preimage attack on Keccak-512

- Keccak-512 ($r = 576$, $c = 1024$, $hash = 512$)
- Message structure: first 8 lanes (512 bits) are unknown (to be determined by the attacker). Last 62 bits of the message are set to 1. The message is padded with two 1's giving a block of 576 bits.
  - It is expected that among 2^{512} possible messages there is, on average, one preimage of a given hash.
  - With such structure, to guess a rotational counterpart of a state we need to care only about 512 unknown bits. (A lane with all 0's or all 1's stays the same before and after rotation.)

3-round preimage attack on Keccak-512

- Keccak-512 ($r = 576$, $c = 1024$, $hash = 512$)
- Message structure: first 8 lanes (512 bits) are unknown (to be determined by the attacker). Last 62 bits of the message are set to 1. The message is padded with two 1's giving a block of 576 bits.
  - It is expected that among 2^{512} possible messages there is, on average, one preimage of a given hash.
  - With such structure, to guess a rotational counterpart of a state we need to care only about 512 unknown bits. (A lane with all 0's or all 1's stays the same before and after rotation.)

3-round preimage attack on Keccak-512

- Keccak-512 ($r = 576$, $c = 1024$, $hash = 512$)
- Message structure: first 8 lanes (512 bits) are unknown (to be determined by the attacker). Last 62 bits of the message are set to 1. The message is padded with two 1's giving a block of 576 bits.
  - It is expected that among 2^{512} possible messages there is, on average, one preimage of a given hash.
  - With such structure, to guess a rotational counterpart of a state we need to care only about 512 unknown bits. (A lane with all 0's or all 1's stays the same before and after rotation.)

3-round preimage attack — main ideas

- The main idea is to find a rotational counterpart of the preimage and show that the workload for this task is below exhaustively trying all 2^{512} values. Once we have a rotational counterpart of the preimage, we simply rotate it back and get the preimage.

3-round preimage attack — main ideas

- Every* 512-bit preimage has 64 possible rotational counterparts (lanes rotated by 1, or lanes rotated by 2, ..., or lanes rotated by 64). Then the probability that we guess one of the rotational counterpart is $2^{-512} \cdot 64 = 2^{-506}$.
- So 2^{506} guesses and we hit a rotational counterpart of the preimage. But how to check which rotational number the guessed rotational counterpart actually has?? (We can not check 64 possibilities as we would end up with 2^{512} — exhaustive search effort)
- Help comes from rotational distinguishers!

3-round preimage attack — main ideas

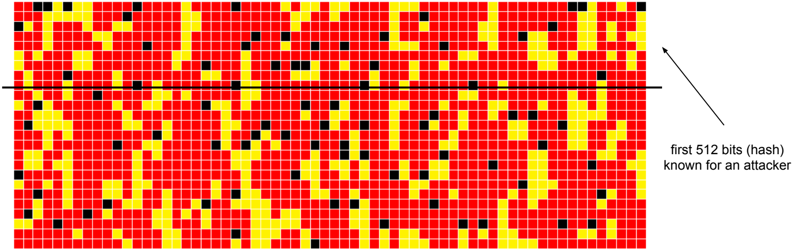
- Every* 512-bit preimage has 64 possible rotational counterparts (lanes rotated by 1, or lanes rotated by 2, ..., or lanes rotated by 64). Then the probability that we guess one of the rotational counterpart is $2^{-512} \cdot 64 = 2^{-506}$.
- So 2^{506} guesses and we hit a rotational counterpart of the preimage. But how to check which rotational number the guessed rotational counterpart actually has?? (We can not check 64 possibilities as we would end up with 2^{512} — exhaustive search effort)
- Help comes from rotational distinguishers!

3-round preimage attack — main ideas

- Every* 512-bit preimage has 64 possible rotational counterparts (lanes rotated by 1, or lanes rotated by 2, ..., or lanes rotated by 64). Then the probability that we guess one of the rotational counterpart is $2^{-512} \cdot 64 = 2^{-506}$.
- So 2^{506} guesses and we hit a rotational counterpart of the preimage. But how to check which rotational number the guessed rotational counterpart actually has?? (We can not check 64 possibilities as we would end up with 2^{512} — exhaustive search effort)
- Help comes from rotational distinguishers!

Use of 3-round distinguishers

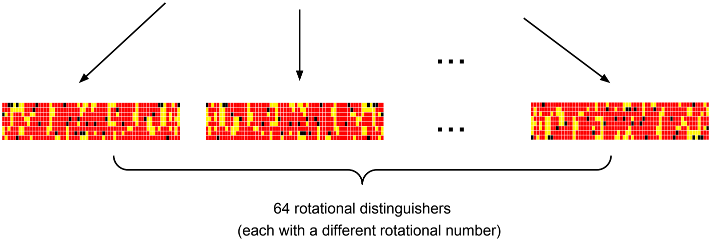
- In precomputation: generate 3-round distinguisher for all 64 rotational numbers. Remember positions of yellow and black squares (in first 512 bits of the state).



- If a guessed preimage is actually the rotational counterpart of the preimage we're looking for, then values of hashes should be as one of the distinguishers shows.

Main loop of the attack

1. guess first 8 lanes (512 bits) of the state, the other bits are fixed according to the structure of the message.
2. run 3-round Keccak-f[1600] on the guessed state.
3. do rotational relations agree with any of 64 distinguishers?



4. **if** (rotational relations agreed) **then** rotate back the guessed state by n bits and run 3-round Keccak-512 on it to check whether the state is the preimage of a given hash.

Complexity of the attack

- The workload of the attack is 2^{256} (checking special messages) + 2^{506} (main loop) + 2^{502} (checking false positive candidates). Thus complexity of the attack is roughly 2^{506} Keccak-512 calls, 64 times better than the exhaustive search.

Extension to 4 rounds

- Direct extension is not possible as there are no yellow or black squares at the end of the 4th round.
- To extend the attack, instead of running Keccak-f[1600] permutation on a guessed state, we run a modified version (without ι step). In consequence, fewer black squares appear and it leads to fewer red, undesirable squares.
- After θ, ρ, and π in the 4th round there are still yellow and black squares (in the first 512 bits). It is good enough for mounting the attack as we can go back to that step from the hash (invert ι and χ from the hash).

Extension to 4 rounds

- Direct extension is not possible as there are no yellow or black squares at the end of the 4th round.
- To extend the attack, instead of running Keccak-f[1600] permutation on a guessed state, we run a modified version (without ι step). In consequence, fewer black squares appear and it leads to fewer red, undesirable squares.
- After θ, ρ, and π in the 4th round there are still yellow and black squares (in the first 512 bits). It is good enough for mounting the attack as we can go back to that step from the hash (invert ι and χ from the hash).

Our attacks and related work

Table: Best known preimage attacks on the Keccak variants proposed as SHA-3 candidates. The number in the column 'Variant' denotes a hash length.

Rounds	Variant	Time	Memory	Reference
6/7/8	512	$2^{506}/2^{507}/2^{511.5}$	$2^{176}/2^{320}/2^{508}$	Bernstein, 2010
4	224/256	$2^{217.3}/2^{249.3}$	2^{61}	Bernstein, 2010
4	384/512	$2^{377.3}/2^{505.3}$	2^{61}	Bernstein, 2010
4	512	2^{506}	negligible	this work
4	384	2^{378}	negligible	this work
4	256	2^{252}	negligible	this work
4	224	2^{221}	negligible	this work

Outline

Conclusion

- Main result: 4-round preimage attack, up to 64 times faster than exhaustive search, negligible amount of memory used in the attack
- 5-round distinguisher on Keccak-f[1600] permutation
- Our attack takes advantage of low Hamming weight of constans. (It would be much harder if there were more 1's in constants.)

Thank you for your attention!