Inverting Step-reduced SHA-1 and MD5 by Parameterized SAT Solvers

Oleg Zaikin

ISDCT SB RAS, Russia

CP 2024

Outline

1. Cryptographic hash functions MD5 and SHA-1
2. Intermediate inverse problems for MD5 and SHA-1
3. SAT encoding
4. Solving intermediate inverse problems by Kissat
5. Tuning Kissat
6. Inverting 24-step SHA-1 and 29-step MD5 by parameterized Kissat

A cryptographic hash function h has the following properties:

1. *Compression*: h maps a message x of arbitrary finite size to a hash $h(x)$ of fixed size.

A cryptographic hash function h has the following properties:

1. *Compression*: h maps a message x of arbitrary finite size to a hash $h(x)$ of fixed size.
2. *Ease of computation*: for any given message x, $h(x)$ is easy to compute.

A cryptographic hash function h has the following properties:

1. *Compression*: h maps a message x of arbitrary finite size to a hash $h(x)$ of fixed size.
2. *Ease of computation*: for any given message x, $h(x)$ is easy to compute.
3. *Preimage resistance*: for any given hash y, it is computationally infeasible to find any of its preimages, i.e. any such message x' that $h(x') = y$.

A cryptographic hash function h has the following properties:

1. *Compression*: h maps a message x of arbitrary finite size to a hash $h(x)$ of fixed size.
2. *Ease of computation*: for any given message x, $h(x)$ is easy to compute.
3. *Preimage resistance*: for any given hash y, it is computationally infeasible to find any of its preimages, i.e. any such message x' that $h(x') = y$.
4. *Second-preimage resistance*: for any given message x, it is computationally infeasible to find x' such that $x' \neq x$, $h(x) = h(x')$.

Cryptographic hash function

A cryptographic hash function h has the following properties:

1. *Compression*: h maps a message x of arbitrary finite size to a hash $h(x)$ of fixed size.
2. *Ease of computation*: for any given message x, $h(x)$ is easy to compute.
3. *Preimage resistance*: for any given hash y, it is computationally infeasible to find any of its preimages, i.e. any such message x' that $h(x') = y$.
4. *Second-preimage resistance*: for any given message x, it is computationally infeasible to find x' such that $x' \neq x, h(x) = h(x')$.
5. *Collision resistance*: it is computationally infeasible to find any two messages x and x' such that $x \neq x', h(x) = h(x')$.

A cryptographic hash function h has the following properties:

1. *Compression*: h maps a message x of arbitrary finite size to a hash $h(x)$ of fixed size.
2. *Ease of computation*: for any given message x, $h(x)$ is easy to compute.
3. *Preimage resistance*: for any given hash y, it is computationally infeasible to find any of its preimages, i.e. any such message x' that $h(x') = y$.
4. *Second-preimage resistance*: for any given message x, it is computationally infeasible to find x' such that $x' \neq x$, $h(x) = h(x')$.
5. *Collision resistance*: it is computationally infeasible to find any two messages x and x' such that $x \neq x'$, $h(x) = h(x')$.

The first two properties are obligatory.

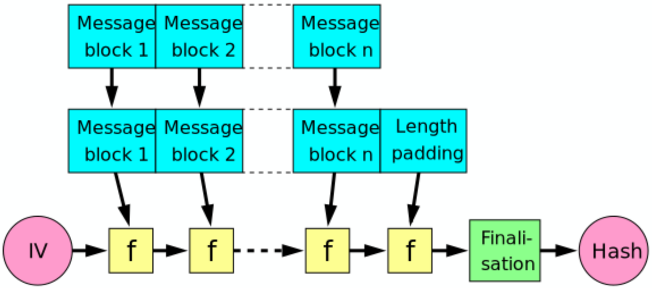
- **Verifying the integrity of messages and files**: compare hashes calculated before and after transmission.

- **Verifying the integrity of messages and files**: compare hashes calculated before and after transmission.
- **Password verification**: they are not stored as clear text, their hashes are stored instead.

- **Verifying the integrity of messages and files**: compare hashes calculated before and after transmission.
- **Password verification**: they are not stored as clear text, their hashes are stored instead.
- **Proof-of-work**: a mining reward is unlocked after some partial hash inversions (e.g. in Bitcoin).

- A method of building cryptographic hash functions from one-way compression functions.

- A method of building cryptographic hash functions from one-way compression functions.

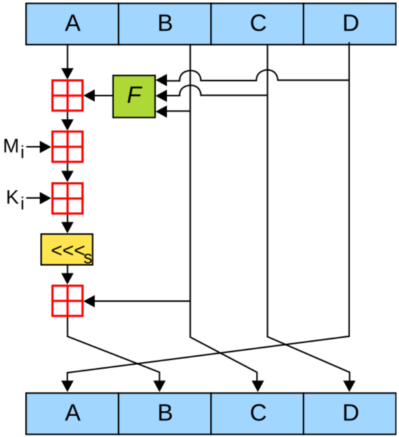


- Initialization vector (IV) has a fixed value.
- The compression function *f* takes the result so far, combines it with a message block, and produces an intermediate result.

- MD5 – a Merkle-Damgard-based cryptographic hash function proposed in 1992.

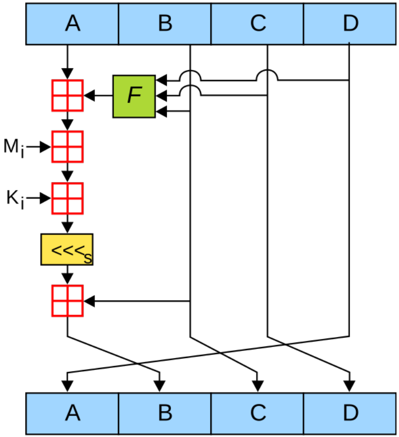
- MD5 – a Merkle-Damgard-based cryptographic hash function proposed in 1992.
- Given a 512-bit message $M = M_0, \ldots, M_{15}$, the compression function produces a 128-bit hash.

- MD5 – a Merkle-Damgard-based cryptographic hash function proposed in 1992.
- Given a 512-bit message $M = M_0, \ldots, M_{15}$, the compression function produces a 128-bit hash.
- Data is transformed in four 32-bit registers A, B, C, D.



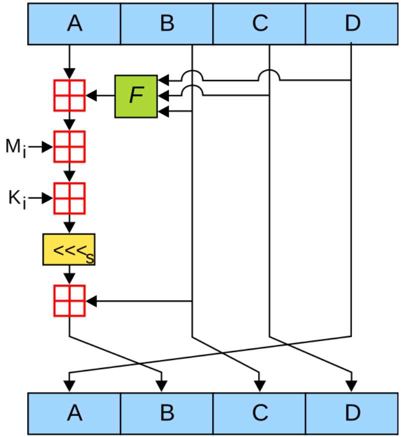
One MD5 step[1]. F is a round function, $\lll$ is circular shift, and $\boxplus$ is addition modulo 2^{32}.

- MD5 – a Merkle-Damgard-based cryptographic hash function proposed in 1992.
- Given a 512-bit message $M = M_0, \ldots, M_{15}$, the compression function produces a 128-bit hash.
- Data is transformed in four 32-bit registers A, B, C, D.
- 64 steps; each step all registers are updated.



One MD5 step[1]. F is a round function, $\lll$ is circular shift, and $\boxplus$ is addition modulo 2^{32}.

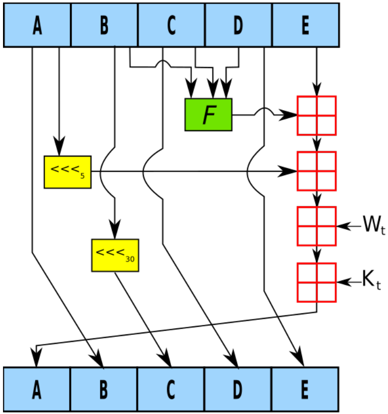
- MD5 – a Merkle-Damgard-based cryptographic hash function proposed in 1992.
- Given a 512-bit message $M = M_0, \ldots, M_{15}$, the compression function produces a 128-bit hash.
- Data is transformed in four 32-bit registers A, B, C, D.
- 64 steps; each step all registers are updated.
- Before the 1st step A, B, C, D are IV, on the last step they are hash.



One MD5 step[1]. F is a round function, $\lll$ is circular shift, and $\boxplus$ is addition modulo 2^{32}.

- SHA-1 – a Merkle-Damgard-based cryptographic hash function proposed in 1995.

- SHA-1 – a Merkle-Damgard-based cryptographic hash function proposed in 1995.
- Compared to MD5:
  1. 160-bit hash.
  2. 5 registers A, B, C, D, E.
  3. 80 steps; 4 rounds, 20 steps each.



One SHA-1 step[2]. F is a round function.

[2] https://en.wikipedia.org/wiki/SHA-1

- Since 2005, MD5 is not collision resistant.
- Since 2017, SHA-1 is not collision resistant.

[3] Florian Legendre et al. Encoding Hash Functions as a SAT Problem // Proc. of ICTAI 2012

MD5 and SHA-1 statuses

- Since 2005, MD5 is not collision resistant.
- Since 2017, SHA-1 is not collision resistant.
- MD5 and SHA-1 are still preimage resistant.
- MD5 and SHA-1 are still used in practice.

[3] Florian Legendre et al. Encoding Hash Functions as a SAT Problem // Proc. of ICTAI 2012

MD5 and SHA-1 statuses

- Since 2005, MD5 is not collision resistant.
- Since 2017, SHA-1 is not collision resistant.
- MD5 and SHA-1 are still preimage resistant.
- MD5 and SHA-1 are still used in practice.
- In 2012, the 28-step MD5 and 23-step SHA-1 were inverted (i.e. their preimages were found) via a SAT solver[3].

[3] Florian Legendre et al. Encoding Hash Functions as a SAT Problem // Proc. of ICTAI 2012

MD5 and SHA-1 statuses

- Since 2005, MD5 is not collision resistant.
- Since 2017, SHA-1 is not collision resistant.
- MD5 and SHA-1 are still preimage resistant.
- MD5 and SHA-1 are still used in practice.
- In 2012, the 28-step MD5 and 23-step SHA-1 were inverted (i.e. their preimages were found) via a SAT solver[3].
- **Goal: invert 29-step MD5 and 24-step SHA-1 via SAT.**

[3] Florian Legendre et al. Encoding Hash Functions as a SAT Problem // Proc. of ICTAI 2012

1. Cryptographic hash functions MD5 and SHA-1
2. **Intermediate inverse problems for MD5 and SHA-1**
3. SAT encoding
4. Solving intermediate inverse problems by Kissat
5. Tuning Kissat
6. Inverting 24-step SHA-1 and 29-step MD5 by parameterized Kissat

The $(i+1)$-th step of MD5:

$$
\begin{aligned}
&temp \leftarrow Func(B, C, D) \boxplus A \boxplus K[i] \boxplus M[g] \\
&A \leftarrow D \\
&D \leftarrow C \\
&C \leftarrow B \\
&B \leftarrow B + (temp \lll s)
\end{aligned}
$$

The $(i+1)$-th step of MD5:

$$temp \leftarrow Func(B, C, D) \boxplus A \boxplus K[i] \boxplus M[g]$$
$$A \leftarrow D$$
$$D \leftarrow C$$
$$C \leftarrow B$$
$$B \leftarrow B + (temp \lll s)$$

Observation: if $M[g]$ is deleted
$temp \leftarrow Func(B, C, D) \boxplus A \boxplus K[i]$
for a CDCL solver the inverse problem is compared to that for i steps:

The $(i+1)$-th step of MD5:

$temp \leftarrow Func(B, C, D) \boxplus A \boxplus K[i] \boxplus M[g]$
$A \leftarrow D$
$D \leftarrow C$
$C \leftarrow B$
$B \leftarrow B + (temp \lll s)$

Observation: if $M[g]$ is deleted
$temp \leftarrow Func(B, C, D) \boxplus A \boxplus K[i]$
for a CDCL solver the inverse problem is compared to that for i steps:

Idea: assign constant values to several bits in $M[g]$ in step $i+1$ thus forming a family of intermediate inverse problems between steps i and $i+1$.

- j is varied from 1 to 31 to form 31 intermediate inverse problems.

- j is varied from 1 to 31 to form 31 intermediate inverse problems.
- $weakM$ is $M[g]$ with $32 - j$ rightmost bits assigned to 0.

Intermediate inverse problems for MD5

- j is varied from 1 to 31 to form 31 intermediate inverse problems.
- *weakM* is $M[g]$ with $32 - j$ rightmost bits assigned to 0.
- Usually $j = 1$ is the simplest (close to inverting i steps), while $j = 31$ is the hardest (close to inverting $i + 1$) steps).

- j is varied from 1 to 31 to form 31 intermediate inverse problems.
- $weakM$ is $M[g]$ with $32 - j$ rightmost bits assigned to 0.
- Usually $j = 1$ is the simplest (close to inverting i steps), while $j = 31$ is the hardest (close to inverting $i + 1$) steps).

A weakened $(i + 1)$-th step of MD5:

$$weakM \leftarrow (M[g] \gg (32 - j)) \ll (32 - j)$$
$$temp \leftarrow Func(B, C, D) \boxplus A \boxplus K[i] \boxplus weakM$$
$$A \leftarrow D$$
$$D \leftarrow C$$
$$C \leftarrow B$$
$$B \leftarrow B + (temp \lll s)$$

- j is varied from 1 to 31 to form 31 intermediate inverse problems.
- $weakM$ is $M[g]$ with $32 - j$ rightmost bits assigned to 0.
- Usually $j = 1$ is the simplest (close to inverting i steps), while $j = 31$ is the hardest (close to inverting $i + 1$) steps).

A weakened $(i + 1)$-th step of MD5:

$$\begin{aligned}
&weakM \leftarrow (M[g] \gg (32 - j)) \ll (32 - j) \\
&temp \leftarrow Func(B, C, D) \boxplus A \boxplus K[i] \boxplus weakM \\
&A \leftarrow D \\
&D \leftarrow C \\
&C \leftarrow B \\
&B \leftarrow B + (temp \lll s)
\end{aligned}$$

- The j-th intermediate MD5 function between i and $i + 1$, is called $(i\ j/32)$-step MD5.

- The main hardness lies in $M[g]$ as well.
- 31 intermediate inverse problems are formed in the same way as for MD5.

1. Cryptographic hash functions MD5 and SHA-1
2. Intermediate inverse problems for MD5 and SHA-1
3. **SAT encoding**
4. Solving intermediate inverse problems by Kissat
5. Tuning Kissat
6. Inverting 24-step SHA-1 and 29-step MD5 by parameterized Kissat

- The best SAT encoding for SHA-1 so far is Vegard Nossum's encoding[4].

[4] Vegard Nossum. SAT-based preimage attacks on SHA-1. Master's thesis, University of Oslo, Department of Informatics, 2012.

[5] `https://github.com/vegard/sha1-sat`

- The best SAT encoding for SHA-1 so far is Vegard Nossum's encoding[4].
- Compared to the competitors, it produces more compact CNFs which are easier for CDCL solvers.

[4] Vegard Nossum. SAT-based preimage attacks on SHA-1. Master's thesis, University of Oslo, Department of Informatics, 2012.

[5] `https://github.com/vegard/sha1-sat`

- The best SAT encoding for SHA-1 so far is Vegard Nossum's encoding[4].
- Compared to the competitors, it produces more compact CNFs which are easier for CDCL solvers.
- The main novelty:addition modulo 2^{32} is expressed as a logic gate circuit and encoded in the clausal form using the ESPRESSO minimizer.

[4] Vegard Nossum. SAT-based preimage attacks on SHA-1. Master's thesis, University of Oslo, Department of Informatics, 2012.

[5] `https://github.com/vegard/sha1-sat`

- The best SAT encoding for SHA-1 so far is Vegard Nossum's encoding[4].
- Compared to the competitors, it produces more compact CNFs which are easier for CDCL solvers.
- The main novelty:addition modulo 2^{32} is expressed as a logic gate circuit and encoded in the clausal form using the ESPRESSO minimizer.
- An implementation is available online[5].

[4] Vegard Nossum. SAT-based preimage attacks on SHA-1. Master's thesis, University of Oslo, Department of Informatics, 2012.

[5] https://github.com/vegard/sha1-sat

- The best SAT encoding for SHA-1 so far is Vegard Nossum's encoding[4].
- Compared to the competitors, it produces more compact CNFs which are easier for CDCL solvers.
- The main novelty:addition modulo 2^{32} is expressed as a logic gate circuit and encoded in the clausal form using the ESPRESSO minimizer.
- An implementation is available online[5].
- In the present study, the implementation is extended to maintain MD5.

[4] Vegard Nossum. SAT-based preimage attacks on SHA-1. Master's thesis, University of Oslo, Department of Informatics, 2012.

[5] `https://github.com/vegard/sha1-sat`

- One cannot assign 0 to $32 - j$ $M[g]$'s bits directly in the CNF since in all previous steps $M[g]$ is used as usual.

SAT encoding of intermediate inverse problems

- One cannot assign 0 to $32 - j$ $M[g]$'s bits directly in the CNF since in all previous steps $M[g]$ is used as usual.
- To encode the j-th intermediate inverse problem between steps i and $i+1$:
  1. 32-bit word weakM is introduced in the form of 32 Boolean variables.
  2. The rightmost $32 - j$ bits of weakM are assigned to 0 via adding unit clauses.
  3. The equality conditions for the leftmost j bits of weakM and the corresponding j bits of $M[g]$ are added in the form of $j \times 2$ binary clauses.
  4. 32 variables of weakW are used instead of $M[g]$'s variables in step $i+1$.

SAT encoding of intermediate inverse problems

- One cannot assign 0 to $32 - j$ $M[g]$'s bits directly in the CNF since in all previous steps $M[g]$ is used as usual.
- To encode the j-th intermediate inverse problem between steps i and $i + 1$:
  1. 32-bit word weakM is introduced in the form of 32 Boolean variables.
  2. The rightmost $32 - j$ bits of weakM are assigned to 0 via adding unit clauses.
  3. The equality conditions for the leftmost j bits of weakM and the corresponding j bits of $M[g]$ are added in the form of $j \times 2$ binary clauses.
  4. 32 variables of weakW are used instead of $M[g]$'s variables in step $i + 1$.

Table: Characteristics of CNFs.

Hash	Steps	Variables	Clauses	Literals
SHA-1	23	4 288	132 672	873 727
SHA-1	23 16/32	4 480	138 812	913 700
SHA-1	24	4 448	138 764	913 620

Outline

1. Cryptographic hash functions MD5 and SHA-1
2. Intermediate inverse problems for MD5 and SHA-1
3. SAT encoding
4. **Solving intermediate inverse problems by Kissat**
5. Tuning Kissat
6. Inverting 24-step SHA-1 and 29-step MD5 by parameterized Kissat

- A state-of-the-art CDCL solver Kissat[6] of version 3.0 is used.

[6] Armin Biere and Mathias Fleury. Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022. In Proc. of SAT Competition 2022.

Solving intermediate inverse problems by Kissat

- A state-of-the-art CDCL solver Kissat[6] of version 3.0 is used.
- 33 inverse problems were considered for SHA-1: 22 steps, 31 intermediate problems between steps 22 and 23, and 23 steps.

[6] Armin Biere and Mathias Fleury. Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022. In Proc. of SAT Competition 2022.

- A state-of-the-art CDCL solver Kissat[6] of version 3.0 is used.
- 33 inverse problems were considered for SHA-1: 22 steps, 31 intermediate problems between steps 22 and 23, and 23 steps.
- For each inverse problem, 10 SAT instances were generated: all 1s hash (1hash), all 0s hash, and 8 random hashes.

[6] Armin Biere and Mathias Fleury. Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022. In Proc. of SAT Competition 2022.

- A state-of-the-art CDCL solver Kissat[6] of version 3.0 is used.
- 33 inverse problems were considered for SHA-1: 22 steps, 31 intermediate problems between steps 22 and 23, and 23 steps.
- For each inverse problem, 10 SAT instances were generated: all 1s hash (1hash), all 0s hash, and 8 random hashes.
- PC with 16-core CPU.

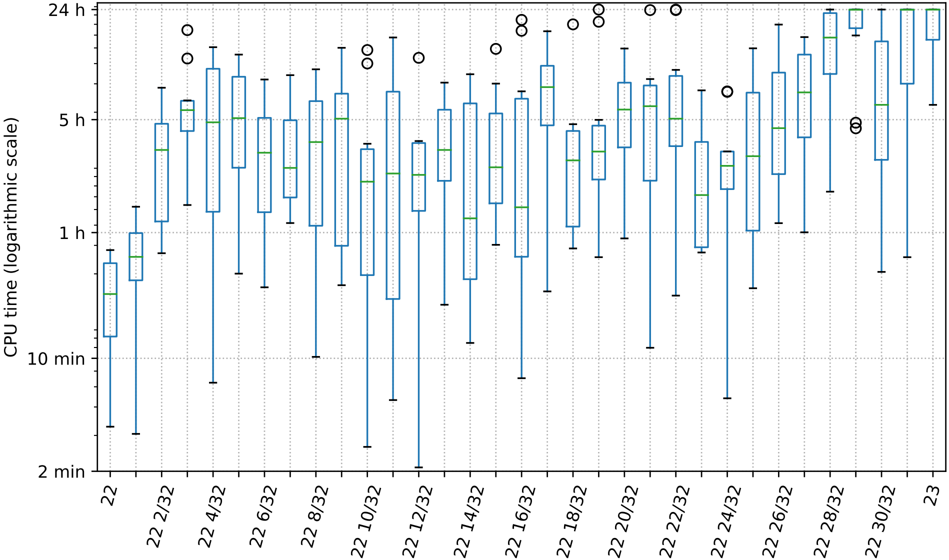
[6] Armin Biere and Mathias Fleury. Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022. In Proc. of SAT Competition 2022.

Solving intermediate inverse problems by Kissat

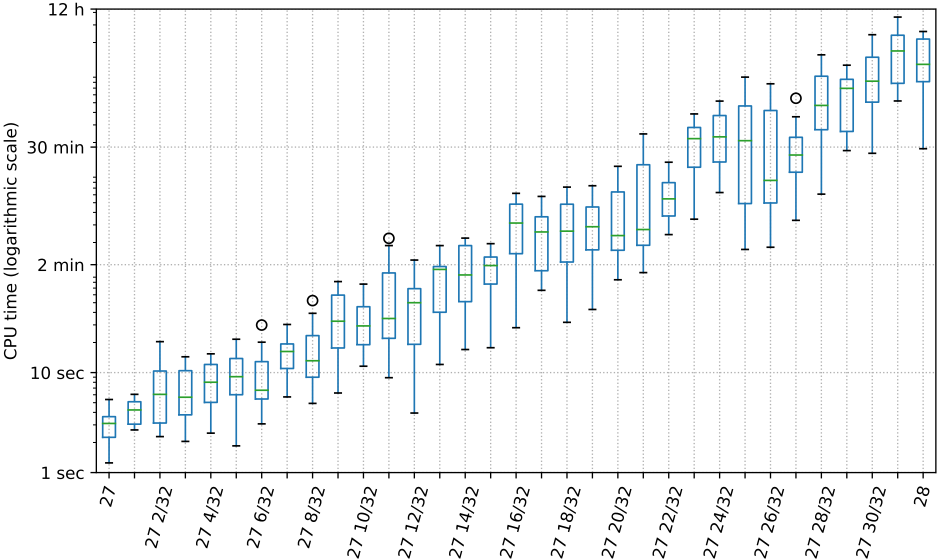
- A state-of-the-art CDCL solver Kissat[6] of version 3.0 is used.
- 33 inverse problems were considered for SHA-1: 22 steps, 31 intermediate problems between steps 22 and 23, and 23 steps.
- For each inverse problem, 10 SAT instances were generated: all 1s hash (1hash), all 0s hash, and 8 random hashes.
- PC with 16-core CPU.
- Time limit 24 hours.

[6] Armin Biere and Mathias Fleury. Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022. In Proc. of SAT Competition 2022.

Boxplots for SHA-1



Boxplots for MD5



Tuning a CDCL solver on intermediate inverse problems

- Kissat has 90 integer parameters.

Tuning a CDCL solver on intermediate inverse problems

- Kissat has 90 integer parameters.
- Kissat can be tuned — one can find a set of parameters' values that minimizes Kissat's total runtime (or PAR2) on a set of SAT instances.

Tuning a CDCL solver on intermediate inverse problems

- Kissat has 90 integer parameters.
- Kissat can be tuned — one can find a set of parameters' values that minimizes Kissat's total runtime (or PAR2) on a set of SAT instances.
- **Problem**: the number of values varies from 2 to millions, so it is infeasible to try all possible sets of parameters' values.

Tuning a CDCL solver on intermediate inverse problems

- Kissat has 90 integer parameters.
- Kissat can be tuned — one can find a set of parameters' values that minimizes Kissat's total runtime (or PAR2) on a set of SAT instances.
- **Problem**: the number of values varies from 2 to millions, so it is infeasible to try all possible sets of parameters' values.
- **Solution**: a metaheuristic algorithm tunes the solver without checking all sets.

Tuning a CDCL solver on intermediate inverse problems

- Kissat has 90 integer parameters.
- Kissat can be tuned — one can find a set of parameters' values that minimizes Kissat's total runtime (or PAR2) on a set of SAT instances.
- **Problem**: the number of values varies from 2 to millions, so it is infeasible to try all possible sets of parameters' values.
- **Solution**: a metaheuristic algorithm tunes the solver without checking all sets.

- **Idea: to invert $i + 1$ steps, tune Kissat on intermediate inverse problems between steps i and $i + 1$.**

- Implementations of metaheuristic algorithms: SMAC3, PyDGGA.

Tuning a CDCL solver on intermediate inverse problems

- Implementations of metaheuristic algorithms: SMAC3, PyDGGA.
- (1+1)-EA (Evolutionary Algorithm) was chosen for tuning because of its simplicity.

Tuning a CDCL solver on intermediate inverse problems

- Implementations of metaheuristic algorithms: SMAC3, PyDGGA.
- (1+1)-EA (Evolutionary Algorithm) was chosen for tuning because of its simplicity.
  1. Consider n parameters.
  2. New set of values: the value of each parameter is changed with probability $1/n$.
  3. Any value can be assigned, but with high probability it will be the closest to the current one.

Outline

1. Cryptographic hash functions MD5 and SHA-1
2. Intermediate inverse problems for MD5 and SHA-1
3. SAT encoding
4. Solving intermediate inverse problems by Kissat
5. Tuning Kissat
6. **Inverting 24-step SHA-1 and 29-step MD5 by parameterized Kissat**

- 16 CNFs in the training set: the last 15 intermediate inverse problems between steps 21-22 and inverting 22-step SHA-1, all for 1-hash.

- 16 CNFs in the training set: the last 15 intermediate inverse problems between steps 21-22 and inverting 22-step SHA-1, all for 1-hash.
- The total runtime on them is **1 hour 58 minutes** on 1 CPU core.

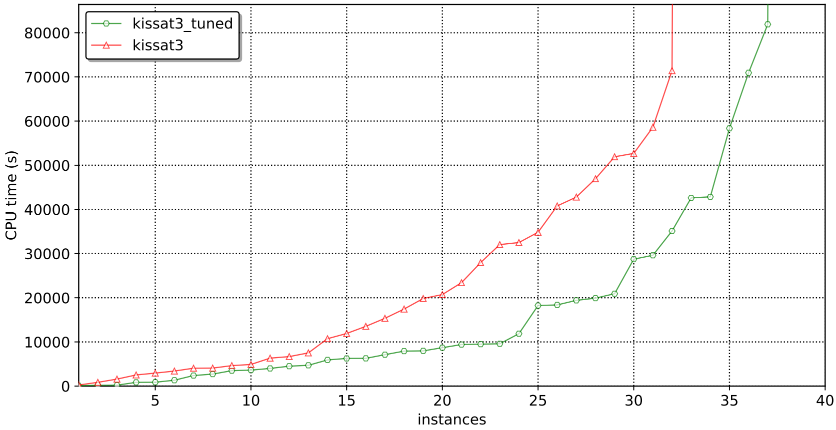
- 16 CNFs in the training set: the last 15 intermediate inverse problems between steps 21-22 and inverting 22-step SHA-1, all for 1-hash.
- The total runtime on them is **1 hour 58 minutes** on 1 CPU core.
- 3 seeds for tuning, each on 16-core CPU during 24 hours.

- 16 CNFs in the training set: the last 15 intermediate inverse problems between steps 21-22 and inverting 22-step SHA-1, all for 1-hash.
- The total runtime on them is **1 hour 58 minutes** on 1 CPU core.
- 3 seeds for tuning, each on 16-core CPU during 24 hours.
- The best set of parameters' values: **22 minutes** in total (5 times faster).

Table: The best KISSAT's configuration found for SHA-1.

Parameter	Default value	Found value
backbonerounds	100	10
definitionticks	1 000 000	100
eliminatebound	16	32
eliminateclslim	100	10
emafast	33	10
minimizedepth	1 000	100
restartmargin	10	20
stable	1	2
sweepfliprounds	1	5
sweepmaxclauses	4 096	2 147 483 647
sweepvars	128	64
vivifytier1	3	2

Tuning Kissat for SHA-1



Comparison of the default KISSAT with its tuned version on intermediate inverse problems for steps 22-24 of SHA-1, 1-hash.

- The Cube-and-Conquer method was applied: a given formula is split via lookahead into a family of simpler subformulas, which are solved by a CDCL solver[7].

[7] Marijn Heule et al. Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads // HVC 2011.

- The Cube-and-Conquer method was applied: a given formula is split via lookahead into a family of simpler subformulas, which are solved by a CDCL solver[7].
- The lookahead solver march_cu split the inverse problem for 24-step SHA-1 into 166 subformulas.

[7] Marijn Heule et al. Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads // HVC 2011.

- The Cube-and-Conquer method was applied: a given formula is split via lookahead into a family of simpler subformulas, which are solved by a CDCL solver[7].
- The lookahead solver march_cu split the inverse problem for 24-step SHA-1 into 166 subformulas.
- The tuned Kissat was run on the subformulas on a supercomputer (166 CPU cores). A preimage was found in 23 hours.

Table: A preimage of 160 1s produced by 24-step SHA-1.

0xa6c5c463	0x182655e0	0x2c5ba5f0	0xe0028033
0x8c3779b1	0x98635880	0xc5b822e	0x297efce7
0x59987038	0xd764eca9	0x7ed9801d	0xdde4f1e0
0x524e678	0xa8ce47dc	0xa813fd76	0x8b58e09f

[7] Marijn Heule et al. Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads // HVC 2011.

- 16 CNFs in the training set: the first 16 intermediate inverse problems between steps 27-28 for 1-hash.

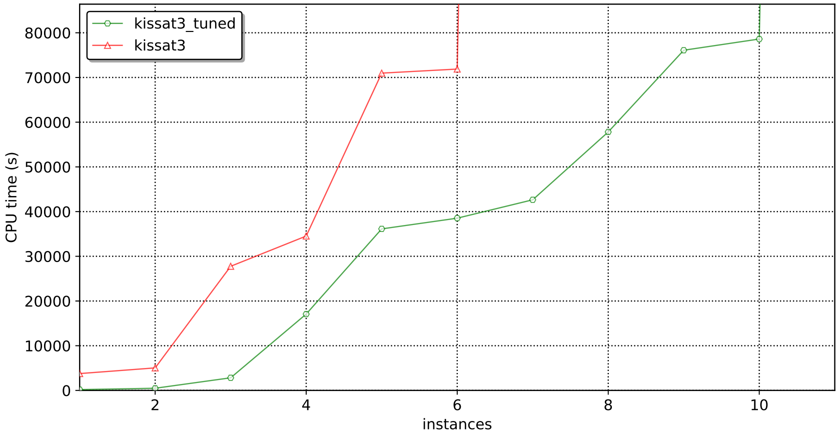
- 16 CNFs in the training set: the first 16 intermediate inverse problems between steps 27-28 for 1-hash.
- The total runtime on them is **14 minutes** on 1 CPU core.

- 16 CNFs in the training set: the first 16 intermediate inverse problems between steps 27-28 for 1-hash.
- The total runtime on them is **14 minutes** on 1 CPU core.
- 3 seeds for tuning, each on 16-core CPU during 24 hours.

- 16 CNFs in the training set: the first 16 intermediate inverse problems between steps 27-28 for 1-hash.
- The total runtime on them is **14 minutes** on 1 CPU core.
- 3 seeds for tuning, each on 16-core CPU during 24 hours.
- The best set of parameters' values: **4 minutes** in total (3 times faster).

Table: The best KISSAT's configuration found for MD5.

Parameter	Default value	Found value
chronolevels	100	1 000
decay	50	32
definitionticks	1 000 000	100
eliminatebound	16	2
eliminateocclim	2 000	1 000
emaslow	100 000	75 000
minimizedepth	1 000	100
restartmargin	10	20
shrink	3	0
stable	1	2
substituterounds	2	32
subsumeclslim	1 000	10 000
sweepmaxclauses	4 096	2 048



Comparison of the default KISSAT with its tuned version on intermediate inverse problems for steps 28-29 of MD5, 1-hash.

- The lookahead solver march_cu split the inverse problem for 29-step MD5 into 74 470 subformulas.

Inverting 29-step MD5

- The lookahead solver march_cu split the inverse problem for 29-step MD5 into 74 470 subformulas.
- The tuned Kissat was run on the subformulas on a supercomputer (540 CPU cores).

- The lookahead solver march_cu split the inverse problem for 29-step MD5 into 74 470 subformulas.
- The tuned Kissat was run on the subformulas on a supercomputer (540 CPU cores).
- A preimage was found in 37 hours.

Table: A preimage of 128 1s produced by 29-step MD5-1.

0xe1051a9e	0x48120773	0x996a5457	0xaaa1d815
0x37d8149c	0x5f999c05	0x182ba14b	0xdfff1673
0xc5db0a2f	0x44430b2a	0xa269f5a2	0x69781b85
0x2b7f0939	0xc1ff3c22	0xc55e990f	0x96ba3fb8

Conclusions

1. A new type of intermediate inverse problems for cryptographic hash functions was proposed.
2. A CDCL solver was tuned on intermediate inverse problems for MD5 and SHA-1.
3. 29-step MD5 and 24-step SHA-1 were inverted for the first time via the tuned solver.
4. In the future, SHA-256 will be studied.

Conclusions

1. A new type of intermediate inverse problems for cryptographic hash functions was proposed.
2. A CDCL solver was tuned on intermediate inverse problems for MD5 and SHA-1.
3. 29-step MD5 and 24-step SHA-1 were inverted for the first time via the tuned solver.
4. In the future, SHA-256 will be studied.

Thank you for your attention! Questions?