Provable insecurity

Where artifacts come from, and how constructive math may help

Claus Diem and dreiwert

University of Leipzig

December 29, 2019

Part I

Problem

Contents

Contents

Signed message

- We would like to have:
  *SHA*3 is collision resistant,
  and therefore GnuPG-SHA3 is unforgeble

Signed message

- We would like to have:
  *SHA*3 is collision resistant,
  and therefore GnuPG-SHA3 is unforgeble
- The problem is:
  What shall “SHA3 is collision resistant” even mean?

What shall “collision resistant” mean?

Computer science guy

- It shall be very hard to find a collision.

What shall “collision resistant” mean?

Computer science guy

- It shall be very hard to find a collision.
- For example: It shall take more that 2^{100} operations.

What shall "collision resistant" mean?

Computer science guy

- It shall be very hard to find a collision.
- For example: It shall take more that 2^{100} operations.
- Key negative example: MD5 is not collision resistant, since collisions can be found within 15 – 30 minutes.

What shall “collision resistant” mean?

Computer science guy

- It shall be very hard to find a collision.
- For example: It shall take more that 2^{100} operations.
- Key negative example: MD5 is not collision resistant, since collisions can be found within 15 – 30 minutes.

What shall “collision resistant” mean?

Computer science guy

- It shall be very hard to find a collision.
- For example: It shall take more that 2^{100} operations.
- Key negative example: MD5 is not collision resistant, since collisions can be found within 15 – 30 minutes.

Math guy

What shall "collision resistant" mean?

Computer science guy

- It shall be very hard to find a collision.
- For example: It shall take more that 2^{100} operations.
- Key negative example: MD5 is not collision resistant, since collisions can be found within 15 – 30 minutes.

Math guy

- For any function h:
  A *collision* is a pair (x, y) with $x \neq y$ and $h(x) = h(y)$

What shall “collision resistant” mean?

Computer science guy

- It shall be very hard to find a collision.
- For example: It shall take more that 2^{100} operations.
- Key negative example: MD5 is not collision resistant, since collisions can be found within 15 – 30 minutes.

Math guy

- For any function h:
  A *collision* is a pair (x, y) with $x \neq y$ and $h(x) = h(y)$
- For a Hash function $h : D \longrightarrow R$ we have $\mathrm{card}(D) > \mathrm{card}(R)$.
- There always *exists* a collision x, y.

What shall “collision resistant” mean?

Computer science guy

- It shall be very hard to find a collision.
- For example: It shall take more that 2^{100} operations.
- Key negative example: MD5 is not collision resistant, since collisions can be found within 15 – 30 minutes.

Math guy

- For any function h:
  A *collision* is a pair (x, y) with $x \neq y$ and $h(x) = h(y)$
- For a Hash function $h : D \longrightarrow R$ we have $\mathrm{card}(D) > \mathrm{card}(R)$.
- There always *exists* a collision x, y.
- So no “real” hash function is collision free.

The math guy’s fastest attack

- ```
  int main() {
    std::cout << "x,y" << std::endl;
    return 0;
  }
  ```
  ```

The math guy's fastest attack

- ```
  int main() {
    std::cout << "x,y" << std::endl;
    return 0;
  }
  ```
- Complexity: constant
  ```

The math guy's fastest attack

- ```
  int main() {
    std::cout << "x,y" << std::endl;
    return 0;
  }
  ```
- Complexity: constant
- The attack always exists
  ```

The math guy's fastest attack

- ```
  int main() {
    std::cout << "x,y" << std::endl;
    return 0;
  }
  ```

- Complexity: constant
- The attack always exists
- Computer science guy: “What!?” You write down an “attack” without knowing the attack?
  ```

The math guy's fastest attack

- ```
  int main() {
    std::cout << "x,y" << std::endl;
    return 0;
  }
  ```

- Complexity: constant
- The attack always exists
- Computer science guy: “What!?” You write down an “attack” without knowing the attack?
- Math guy: “Yes, it exists” ...
  ```

What shall “collision resistant” mean?

Theoretical cryptographer

What shall “collision resistant” mean?

Theoretical cryptographer

What shall “collision resistant” mean?

Theoretical cryptographer

- The mathematician is right, but the conclusion is not acceptable.

What shall “collision resistant” mean?

Theoretical cryptographer

- The mathematician is right, but the conclusion is not acceptable.
- Therefore, we introduce a *parameter* and look at it from an *asymptotic point of view*.

What shall “collision resistant” mean?

Theoretical cryptographer

- The mathematician is right, but the conclusion is not acceptable.
- Therefore, we introduce a *parameter* and look at it from an *asymptotic point of view*.
- We look at attackers running in *polynomial time*, talk about *success probability*.

What shall “collision resistant” mean?

Theoretical cryptographer

- The mathematician is right, but the conclusion is not acceptable.
- Therefore, we introduce a *parameter* and look at it from an *asymptotic point of view*.
- We look at attackers running in *polynomial time*, talk about *success probability*.
- And then later we fix the parameter and apply this to a “real” system.

Variable output length

- We have $h = (h_s)_s$ with $h_s : \{0,1\}^* \to \{0,1\}^{\ell(s)}$ (security parameter s)

Variable output length

- We have $h = (h_s)_s$ with $h_s : \{0,1\}^* \to \{0,1\}^{\ell(s)}$ (security parameter s)
- Attacker A gets $1^{\ell(s)}$ as an input, outputs x, y

Variable output length

- We have $h = (h_s)_s$ with $h_s : \{0,1\}^* \to \{0,1\}^{\ell(s)}$ (security parameter s)
- Attacker A gets $1^{\ell(s)}$ as an input, outputs x, y
- Collision resistance: $\forall n : \exists s_0 : \forall s : s > s_0 \Rightarrow P[x \neq y \wedge h_s(x) = h_s(y)] < \frac{1}{\ell(s)^n}$

Variable output length

- We have $h = (h_s)_s$ with $h_s : \{0,1\}^* \to \{0,1\}^{\ell(s)}$ (security parameter s)
- Attacker A gets $1^{\ell(s)}$ as an input, outputs x, y
- Collision resistance: $\forall n : \exists s_0 : \forall s : s > s_0 \Rightarrow P[x \neq y \wedge h_s(x) = h_s(y)] < \frac{1}{\ell(s)^n}$
- (after Rogaway, 2007)

Artifact: ℓ

- Suppose the family $h = (h_s)_s$ is collision free.
  What can we then conclude about h_{s_0} for a *particular* paramater s_0?

Artifact: ℓ

- Suppose the family $h = (h_s)_s$ is collision free.
  What can we then conclude about h_{s_0} for a *particular* paramater s_0?
- Strictly speaking nothing:

Artifact: ℓ

- Suppose the family $h = (h_s)_s$ is collision free.
  What can we then conclude about h_{s_0} for a *particular* paramater s_0?
- Strictly speaking nothing:
- Suppose h is collision resistant and $h^*_s = \begin{cases} h_s, \text{ if } l(s) \neq 128, \\ MD5, \text{ if } l(s) = 128. \end{cases}$
  Then h^* is also collision resistant by the definition.

Artifact: ℓ

- Suppose the family $h = (h_s)_s$ is collision free.
  What can we then conclude about h_{s_0} for a *particular* paramater s_0?
- Strictly speaking nothing:
- Suppose h is collision resistant and $h_s^* = \begin{cases} h_s, \text{ if } l(s) \neq 128, \\ MD5, \text{ if } l(s) = 128. \end{cases}$
  Then h^* is also collision resistant by the definition.
- But MD5 is still broken ...

Artifact: ℓ

- Suppose the family $h = (h_s)_s$ is collision free.
  What can we then conclude about h_{s_0} for a *particular* paramater s_0?
- Strictly speaking nothing:
- Suppose h is collision resistant and $h^*_s = \begin{cases} h_s, \text{ if } l(s) \neq 128, \\ MD5, \text{ if } l(s) = 128. \end{cases}$
  Then h^* is also collision resistant by the definition.
- But MD5 is still broken ...
- Such a family h^* might seem to be “artificially constructed”, but maybe not ...

Keyed hash functions

- $h_{s,k} : \{0,1\}^* \to \{0,1\}^{l(s)}$ (security parameter s, key k)

Keyed hash functions

- $h_{s,k} : \{0,1\}^* \to \{0,1\}^{l(s)}$ (security parameter s, key k)
- Attacker A_s reads k, outputs x, y

Keyed hash functions

- $h_{s,k} : \{0,1\}^* \to \{0,1\}^{l(s)}$ (security parameter s, key k)
- Attacker A_s reads k, outputs x, y
- collision resistant: $\forall n : \exists s_0 : \forall s : s > s_0 \Rightarrow P[x \neq y \wedge h_{s,k}(x) = h_{s,k}(y)] < \frac{1}{l(s)^n}$

Keyed hash functions

- $h_{s,k} : \{0,1\}^* \to \{0,1\}^{l(s)}$ (security parameter s, key k)
- Attacker A_s reads k, outputs x, y
- collision resistant: $\forall n : \exists s_0 : \forall s : s > s_0 \Rightarrow P[x \neq y \wedge h_{s,k}(x) = h_{s,k}(y)] < \frac{1}{l(s)^n}$
- (after Damgard 1987)

Keyed hash functions

- $h_{s,k} : \{0,1\}^* \to \{0,1\}^{l(s)}$ (security parameter s, key k)
- Attacker A_s reads k, outputs x, y
- collision resistant: $\forall n : \exists s_0 : \forall s : s > s_0 \Rightarrow P[x \neq y \wedge h_{s,k}(x) = h_{s,k}(y)] < \frac{1}{l(s)^n}$
- (after Damgard 1987)
- Allows working with A_s working on fixed output lengths

Keyed hash functions

- $h_{s,k} : \{0,1\}^* \to \{0,1\}^{l(s)}$ (security parameter s, key k)
- Attacker A_s reads k, outputs x, y
- collision resistant: $\forall n : \exists s_0 : \forall s : s > s_0 \Rightarrow P[x \neq y \wedge h_{s,k}(x) = h_{s,k}(y)] < \frac{1}{l(s)^n}$
- (after Damgard 1987)
- Allows working with A_s working on fixed output lengths
- Might seem to be a good solution: Not asymptotic, does not immediately lead to a “trivial” attack.

Artifact: k

- But: Real hash functions normally don't have keys

Artifact: k

- But: Real hash functions normally don't have keys
- Possible interpretation in some cases: key = initialization vector

Artifact: k

- But: Real hash functions normally don't have keys
- Possible interpretation in some cases: key = initialization vector
- But then, free-start collision attacks are being analyzed

Artifact: k

- But: Real hash functions normally don't have keys
- Possible interpretation in some cases: key = initialization vector
- But then, free-start collision attacks are being analyzed
- But without variable (!) k, A_s can always be the trivial attacker

Artifact: k

- But: Real hash functions normally don't have keys
- Possible interpretation in some cases: key = initialization vector
- But then, free-start collision attacks are being analyzed
- But without variable (!) k, A_s can always be the trivial attacker
- Assume h being collision resistant and

$$h^*_{s,k} = \begin{cases} h_{s,k}, \text{ if } l(s) \neq 128, \\ MD5, \text{ if } l(s) = 128 \wedge k = k_0, \end{cases}$$

Artifact: k

- But: Real hash functions normally don't have keys
- Possible interpretation in some cases: key = initialization vector
- But then, free-start collision attacks are being analyzed
- But without variable (!) k, A_s can always be the trivial attacker
- Assume h being collision resistant and

$$h^*_{s,k} = \begin{cases} h_{s,k}, \text{ if } l(s) \neq 128, \\ MD5, \text{ if } l(s) = 128 \wedge k = k_0, \end{cases}$$

- So, strictly speaking from "h is collision resistant" we still cannot conclude anything about "concrete hash functions".

Practical security

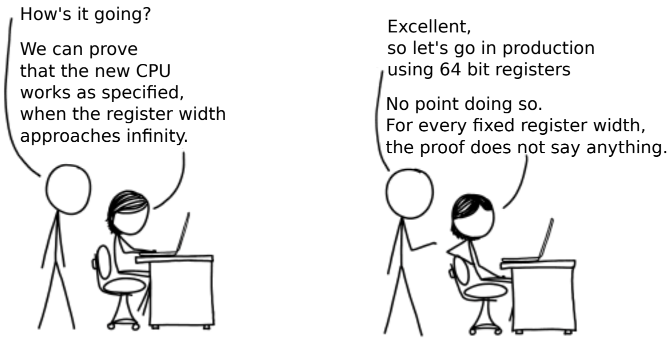

Figure: Drawings: xkcd.com, modification to text (CC BY-NC 2.5)

“Provably secure” hash functions

- collision resistant hash functions according to these definitions can be constructed

“Provably secure” hash functions

- collision resistant hash functions according to these definitions can be constructed (under suitable assumption!).
- e.g. VSH, ECOH, FSB

“Provably secure” hash functions

- collision resistant hash functions according to these definitions can be constructed (under suitable assumption!).
- e.g. VSH, ECOH, FSB
- Often slow and of little practical relevance

"Provably secure" hash functions

- collision resistant hash functions according to these definitions can be constructed (under suitable assumption!).
- e.g. VSH, ECOH, FSB
- Often slow and of little practical relevance
- Who decides about the length and the key to use?

First conclusions

- Problematic to characterize families of functions when seeking for results on a specific hash functions

First conclusions

- Problematic to characterize families of functions when seeking for results on a specific hash functions
- Where does the (existing) attacker A come from?

First conclusions

- Problematic to characterize families of functions when seeking for results on a specific hash functions
- Where does the (existing) attacker A come from?
- Explicit precomputation: A_{pre} computes attacker A

First conclusions

- Problematic to characterize families of functions when seeking for results on a specific hash functions
- Where does the (existing) attacker A come from?
- Explicit precomputation: A_{pre} computes attacker A
- Cost of attack: e.g. $TIME(A_{pre}) + TIME(A)$

The fastest attack, reloaded

- ```
  int main() {
    std::cout << "int main() {" << std::endl;
    std::cout << "  std::cout << \"x,y\\n\";\n";
    std::cout << "  return 0;" << std::endl;
    std::cout << "}" << std::endl;
    return 0;
  }
  ```
  ```

The fastest attack, reloaded

- ```
  int main() {
    std::cout << "int main() {" << std::endl;
    std::cout << "  std::cout << \"x,y\\n\";\n";
    std::cout << "  return 0;" << std::endl;
    std::cout << "}" << std::endl;
    return 0;
  }
  ```
- Complexity: constant
  ```

The fastest attack, reloaded

- ```
  int main() {
    std::cout << "int main() {" << std::endl;
    std::cout << "  std::cout << \"x,y\\n\";\n";
    std::cout << "  return 0;" << std::endl;
    std::cout << "}" << std::endl;
    return 0;
  }
  ```
- Complexity: constant
- Anything gained?
  ```

Closing the gap

- An idea (after Bernstein and Lange 2012):
  Size limitation for A_{pre}

Closing the gap

- An idea (after Bernstein and Lange 2012):
  Size limitation for A_{pre}
- Outrules trivial attacks for sufficiently large output lengths

Closing the gap

- An idea (after Bernstein and Lange 2012):
  Size limitation for A_{pre}
- Outrules trivial attacks for sufficiently large output lengths
- Still not useful for practically used hash functions.

Fundamental issue remains

- We know: If a Hash function *h* is collision resistant GnuPG-h is unforgable.

Fundamental issue remains

- We know: If a Hash function h is collision resistant GnuPG-h is unforgable.
- We want to argue that some “real” Hash function h is collision resistant.

Fundamental issue remains

- We know: If a Hash function h is collision resistant GnuPG-h is unforgable.
- We want to argue that some “real” Hash function h is collision resistant.
- But such an h is *never* collision resistant.

Fundamental issue remains

- We know: If a Hash function h is collision resistant GnuPG-h is unforgable.
- We want to argue that some “real” Hash function h is collision resistant.
- But such an h is *never* collision resistant.
- Only in the asymptotic setting or in the Random Oracle model this can be proven.

Fundamental issue remains

- We know: If a Hash function *h* is collision resistant GnuPG-h is unforgable.
- We want to argue that some “real” Hash function *h* is collision resistant.
- But such an *h* is *never* collision resistant.
- Only in the asymptotic setting or in the Random Oracle model this can be proven.
- So usually the known proofs are applied where they cannot really be applied
- Is this really what we expect from a „proof“?

Interpretation of proofs

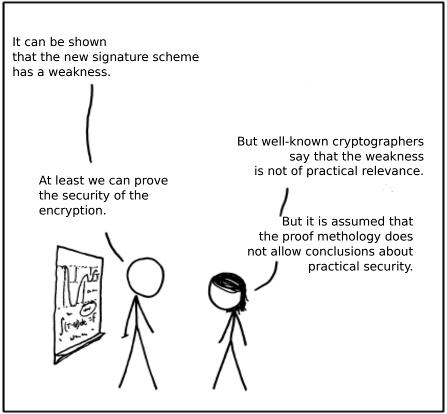

Figure: Drawings: xkcd.com, modification to text (CC BY-NC 2.5)

Getting to the root cause

- Where do x and y come from?

Getting to the root cause

- Where do x and y come from?
- $x, y \leftarrow$ pigeonhole principle $\leftarrow$ mathematical logic

Getting to the root cause

- Where do x and y come from?
- $x, y \leftarrow$ pigeonhole principle $\leftarrow$ mathematical logic
- Language consisting of: $\vee$, $\wedge$, $\neg$, $\implies$, $\exists$, $\forall$ and symbols

Getting to the root cause

- Where do x and y come from?
- $x, y \leftarrow$ pigeonhole principle $\leftarrow$ mathematical logic
- Language consisting of: $\vee$, $\wedge$, $\neg$, $\implies$, $\exists$, $\forall$ and symbols
- Problem may be caused by the meaning of the symbols

Part II

Constructive logic

What is constructive logic?

- Symbols as in classical logic

What is constructive logic?

- Symbols as in classical logic
- Meaning partially different

What is constructive logic?

- Symbols as in classical logic
- Meaning partially different
- “x exists” means “we can construct x”

From proofs to algorithms

- BHK interpretations give a meaning to constructive proofs.

From proofs to algorithms

- BHK interpretations give a meaning to constructive proofs.
- (after Brouwer-Heyting-Kolmogorov, more seldomly Brouwer-Heyting-Kreisel)

From proofs to algorithms

- BHK interpretations give a meaning to constructive proofs.
- (after Brouwer-Heyting-Kolmogorov, more seldomly Brouwer-Heyting-Kreisel)
- *Realizations* formalize these interpretations.

From proofs to algorithms

- BHK interpretations give a meaning to constructive proofs.
- (after Brouwer-Heyting-Kolmogorov, more seldomly Brouwer-Heyting-Kreisel)
- *Realizations* formalize these interpretations.
- Realizations have a strong relationship to algorithms

What are realizations?

- “a realizes A” means:

What are realizations?

- "a realizes A" means: a is a proof of A
- defined inductively over the structure of the proven formula

Conjunction

- structure: $A \wedge B$

Conjunction

- structure: $A \wedge B$
- $\langle a, b \rangle$ realizes $A \wedge B$ iff a realizes A and b realizes B

Conjunction

- structure: $A \wedge B$
- $\langle a, b \rangle$ realizes $A \wedge B$ iff a realizes A and b realizes B
- Interpretation: both conjuncts must be proved

Conjunction

- structure: $A \wedge B$
- $\langle a, b \rangle$ realizes $A \wedge B$ iff a realizes A and b realizes B
- Interpretation: both conjuncts must be proved
- Meaning as in classical logic

Disjunction

- structure: $A \vee B$

Disjunction

- structure: $A \vee B$
- $\langle 0, a \rangle$ realizes $A \vee B$ iff a realizes A
- $\langle 1, b \rangle$ realizes $A \vee B$ iff b realizes B

Disjunction

- structure: $A \vee B$
- $\langle 0, a \rangle$ realizes $A \vee B$ iff a realizes A
- $\langle 1, b \rangle$ realizes $A \vee B$ iff b realizes B
- Interpretation: one must either prove A or prove B

Disjunction

- structure: $A \vee B$
- $\langle 0, a \rangle$ realizes $A \vee B$ iff a realizes A
- $\langle 1, b \rangle$ realizes $A \vee B$ iff b realizes B
- Interpretation: one must either prove A or prove B
- Stronger meaning as a disjunction in classical logic

Implication

- structure: $A \Rightarrow B$

Implication

- structure: $A \Rightarrow B$
- f realizes $A \Rightarrow B$ means: If a realizes A then $f(a)$ realizes B

Implication

- structure: $A \Rightarrow B$
- f realizes $A \Rightarrow B$ means: If a realizes A then $f(a)$ realizes B
- Interpretation: convert any proof for A into a proof for B

Implication

- structure: $A \Rightarrow B$
- f realizes $A \Rightarrow B$ means: If a realizes A then $f(a)$ realizes B
- Interpretation: convert any proof for A into a proof for B
- Meaning as in classical logic

Negation

- structure: $\neg A$

Negation

- structure: $\neg A$
- f realizes $\neg A$ iff. f realizes $A \Rightarrow 0 = 1$

Negation

- structure: $\neg A$
- f realizes $\neg A$ iff. f realizes $A \Rightarrow 0 = 1$
- Interpretation: derive a contradiction from any proof for A

Negation

- structure: $\neg A$
- f realizes $\neg A$ iff. f realizes $A \Rightarrow 0 = 1$
- Interpretation: derive a contradiction from any proof for A
- Meaning weaker as a negation in classical logic

Negation

- structure: $\neg A$
- f realizes $\neg A$ iff. f realizes $A \Rightarrow 0 = 1$
- Interpretation: derive a contradiction from any proof for A
- Meaning weaker as a negation in classical logic
- $A \Rightarrow \neg\neg A$, but not necessarily $\neg\neg A \Rightarrow A$

Universal quantification

- structure: $\forall x : A$

Universal quantification

- structure: $\forall x : A$
- f realizes $\forall x : A$ iff. $f(a)$ realizes $A[x/a]$ for every a

Universal quantification

- structure: $\forall x : A$
- f realizes $\forall x : A$ iff. $f(a)$ realizes $A[x/a]$ for every a
- Interpretation: convert any object a into a proof for $A[x/a]$

Universal quantification

- structure: $\forall x : A$
- f realizes $\forall x : A$ iff. $f(a)$ realizes $A[x/a]$ for every a
- Interpretation: convert any object a into a proof for $A[x/a]$
- Meaning as in classical logic

Existential quantification

- structure: $\exists x : A$

Existential quantification

- structure: $\exists x : A$
- $\langle w, a \rangle$ realizes $\exists x : A$ iff. a realizes $A[x / w]$

Existential quantification

- structure: $\exists x : A$
- $\langle w, a\rangle$ realizes $\exists x : A$ iff. a realizes $A[x/w]$
- Interpretation: name a witness w, and prove that $A[x/w]$ holds

Existential quantification

- structure: $\exists x : A$
- $\langle w, a \rangle$ realizes $\exists x : A$ iff. a realizes $A[x/w]$
- Interpretation: name a witness w, and prove that $A[x/w]$ holds
- Stronger meaning as an existential quantification in classical logic

Lambda expressions

- Lambda expressions as a representation of realizations

Lambda expressions

- Lambda expressions as a representation of realizations
- Lambda expressions Λ over a set of variables $\mathbb{L}$ are:

Lambda expressions

- Lambda expressions as a representation of realizations
- Lambda expressions Λ over a set of variables $\mathbb{L}$ are:
- Variables l where $l \in \mathbb{L}$

Lambda expressions

- Lambda expressions as a representation of realizations
- Lambda expressions Λ over a set of variables $\mathbb{L}$ are:
- Variables l where $l \in \mathbb{L}$
- Applications AB where $\{A, B\} \subset \Lambda$

Lambda expressions

- Lambda expressions as a representation of realizations
- Lambda expressions Λ over a set of variables $\mathbb{L}$ are:
- Variables I where $I \in \mathbb{L}$
- Applications AB where $\{A, B\} \subset \Lambda$
- Abstractions $\lambda x : A$ where $x \in \mathbb{L}$ and $A \in \Lambda$

Lambda calculus

- Lambda calculus on lambda expressions through beta reduction

Lambda calculus

- Lambda calculus on lambda expressions through beta reduction
- $(\lambda x : A)B \underset{\beta}{\rightarrow} A[x/B]$ (A, where occurrences of x are substituted by B)

Lambda calculus

- Lambda calculus on lambda expressions through beta reduction
- $(\lambda x : A)B \underset{\beta}{\rightarrow} A[x/B]$ (A, where occurrences of x are substituted by B)
- $AB \underset{\beta}{\rightarrow} AC$, where $B \underset{\beta}{\rightarrow} C$
- $AC \underset{\beta}{\rightarrow} BC$, where $A \underset{\beta}{\rightarrow} B$

Lambda calculus

- Lambda calculus on lambda expressions through beta reduction
- $(\lambda x : A)B \underset{\beta}{\rightarrow} A[x/B]$ (A, where occurrences of x are substituted by B)
- $AB \underset{\beta}{\rightarrow} AC$, where $B \underset{\beta}{\rightarrow} C$
- $AC \underset{\beta}{\rightarrow} BC$, where $A \underset{\beta}{\rightarrow} B$
- Turing complete (Church-Turing-thesis)

Lambda calculus

- Lambda calculus on lambda expressions through beta reduction
- $(\lambda x : A)B \underset{\beta}{\rightarrow} A[x/B]$ (A, where occurrences of x are substituted by B)
- $AB \underset{\beta}{\rightarrow} AC$, where $B \underset{\beta}{\rightarrow} C$
- $AC \underset{\beta}{\rightarrow} BC$, where $A \underset{\beta}{\rightarrow} B$
- Turing complete (Church-Turing-thesis)
- Example: $(\lambda x : 2(x+y))3 \underset{\beta}{\rightarrow} 2(3+y)$

Lambda calculus

- Lambda calculus on lambda expressions through beta reduction
- $(\lambda x : A)B \underset{\beta}{\rightarrow} A[x/B]$ (A, where occurrences of x are substituted by B)
- $AB \underset{\beta}{\rightarrow} AC$, where $B \underset{\beta}{\rightarrow} C$
- $AC \underset{\beta}{\rightarrow} BC$, where $A \underset{\beta}{\rightarrow} B$
- Turing complete (Church-Turing-thesis)
- Example: $(\lambda x : 2(x + y))3 \underset{\beta}{\rightarrow} 2(3 + y)$
- Counting beta reductions can lead to a time complexity measure

Emulating classical logic

- The behaviour of classical logic can achieved by working with formulas in negative form

Emulating classical logic

- The behaviour of classical logic can achieved by working with formulas in negative form
- $\neg\forall x : \neg A$ instead of $\exists x : A$

Emulating classical logic

- The behaviour of classical logic can achieved by working with formulas in negative form
- $\neg\forall x : \neg A$ instead of $\exists x : A$
- $\neg(\neg A \wedge \neg B)$ instead of $A \vee B$

Emulating classical logic

- The behaviour of classical logic can achieved by working with formulas in negative form
- $\neg\forall x : \neg A$ instead of $\exists x : A$
- $\neg(\neg A \wedge \neg B)$ instead of $A \vee B$
- $\neg\neg A$ instead of A

Emulating classical logic

- The behaviour of classical logic can achieved by working with formulas in negative form
- $\neg\forall x : \neg A$ instead of $\exists x : A$
- $\neg(\neg A \wedge \neg B)$ instead of $A \vee B$
- $\neg\neg A$ instead of A
- On these, classical rules of inference apply

Algorithmic content

- $\langle a, b \rangle$ realizes $A \land B$

Algorithmic content

- $\langle a, b \rangle$ realizes $A \wedge B$
- $\langle v, a \rangle$ realizes $A \vee B$

Algorithmic content

- $\langle a, b \rangle$ realizes $A \wedge B$
- $\langle v, a \rangle$ realizes $A \vee B$
- f realizes $A \Rightarrow B$

Algorithmic content

- $\langle a, b \rangle$ realizes $A \wedge B$
- $\langle v, a \rangle$ realizes $A \vee B$
- f realizes $A \Rightarrow B$
- f realizes $\forall x : A$

Algorithmic content

- $\langle a, b \rangle$ realizes $A \wedge B$
- $\langle v, a \rangle$ realizes $A \vee B$
- f realizes $A \Rightarrow B$
- f realizes $\forall x : A$
- $\langle w, a \rangle$ realizes $\exists x : A$

Algorithmic content

- $\langle a, b\rangle$ realizes $A \wedge B$
- $\langle v, a\rangle$ realizes $A \vee B$
- f realizes $A \Rightarrow B$
- f realizes $\forall x : A$
- $\langle w, a\rangle$ realizes $\exists x : A$
- Algorithms can be extracted from the realization of „positive“ formulas

Law of excluded middle

- $A \vee \neg A$ does not hold in general

Law of excluded middle

- $A \vee \neg A$ does not hold in general
- For specific A, it may be provable

Law of excluded middle

- $A \vee \neg A$ does not hold in general
- For specific A, it may be provable
- Thus, lemmas are often of the form $\forall xyz \ldots : P(x, y, z, \ldots) \vee \neg P(x, y, z, \ldots)$

Law of excluded middle

- $A \vee \neg A$ does not hold in general
- For specific A, it may be provable
- Thus, lemmas are often of the form $\forall xyz\ldots : P(x, y, z, \ldots) \vee \neg P(x, y, z, \ldots)$
- e.g. $\forall xy : (x = y) \vee \neg(x = y)$

Law of excluded middle

- $A \vee \neg A$ does not hold in general
- For specific A, it may be provable
- Thus, lemmas are often of the form $\forall xyz\ldots : P(x, y, z, \ldots) \vee \neg P(x, y, z, \ldots)$
- e.g. $\forall xy : (x = y) \vee \neg(x = y)$
- Realization $f(x, y) = \begin{cases} \langle 0, a \rangle, \text{ if } x = y, \\ \langle 1, b \rangle, \text{ if } x \neq y. \end{cases}$

Law of excluded middle

- $A \vee \neg A$ does not hold in general
- For specific A, it may be provable
- Thus, lemmas are often of the form $\forall xyz\ldots : P(x, y, z, \ldots) \vee \neg P(x, y, z, \ldots)$
- e.g. $\forall xy : (x = y) \vee \neg(x = y)$
- Realization $f(x, y) = \begin{cases} \langle 0, a \rangle, \text{ if } x = y, \\ \langle 1, b \rangle, \text{ if } x \neq y. \end{cases}$
- In extracted algorithms: „subroutine“

Constructive math

- Only pure logic considered so far

Constructive math

- Only pure logic considered so far
- To define mathematical objects, axioms are needed

Constructive math

- Only pure logic considered so far
- To define mathematical objects, axioms are needed
- Important for algorithmic content: mathematical induction

Induction

- $\forall P : (P(0) \wedge \forall n : P(n) \Rightarrow P(n+1)) \Rightarrow \forall n : P(n)$

Induction

- $\forall P : (P(0) \wedge \forall n : P(n) \Rightarrow P(n+1)) \Rightarrow \forall n : P(n)$
- An „interface“ for the realization is given by this structure

Induction

- $\forall P : (P(0) \wedge \forall n : P(n) \Rightarrow P(n+1)) \Rightarrow \forall n : P(n)$
- An „interface“ for the realization is given by this structure
- $IP\langle A, \lambda n : B\rangle n$ (A base case, B induction step)

Induction

- $\forall P : (P(0) \wedge \forall n : P(n) \Rightarrow P(n+1)) \Rightarrow \forall n : P(n)$
- An „interface“ for the realization is given by this structure
- $IP\langle A, \lambda n : B\rangle n$ (A base case, B induction step)
- extracted algorithm: recursive

Hash collision as a positive formula

- $\exists A : P[A(r) = \langle x, y \rangle \wedge x \neq y \wedge h(x) = h(y)] > \varepsilon$ (r source of randomness)

Hash collision as a positive formula

- $\exists A : P[A(r) = \langle x, y \rangle \wedge x \neq y \wedge h(x) = h(y)] > \varepsilon$ (r source of randomness)
- or: $\exists A : \neg(P[A(r) = \langle x, y \rangle \wedge x \neq y \wedge h(x) = h(y)] \leq \varepsilon)$

Hash collision as a positive formula

- $\exists A : P[A(r) = \langle x, y\rangle \wedge x \neq y \wedge h(x) = h(y)] > \varepsilon$ (r source of randomness)
- or: $\exists A : \neg(P[A(r) = \langle x, y\rangle \wedge x \neq y \wedge h(x) = h(y)] \leq \varepsilon)$
- A_{pre} is the algorithm extracted from the realization

Hash collision as a positive formula

- $\exists A : P[A(r) = \langle x, y\rangle \wedge x \neq y \wedge h(x) = h(y)] > \varepsilon$ (r source of randomness)
- or: $\exists A : \neg(P[A(r) = \langle x, y\rangle \wedge x \neq y \wedge h(x) = h(y)] \leq \varepsilon)$
- A_{pre} is the algorithm extracted from the realization
- Where a collision x, y is known, the realization can be written as $\langle \lambda r : \langle x, y\rangle, a\rangle$ (a having no algorithmic content)

Hash collision as a positive formula

- $\exists A : P[A(r) = \langle x, y\rangle \wedge x \neq y \wedge h(x) = h(y)] > \varepsilon$ (r source of randomness)
- or: $\exists A : \neg(P[A(r) = \langle x, y\rangle \wedge x \neq y \wedge h(x) = h(y)] \leq \varepsilon)$
- A_{pre} is the algorithm extracted from the realization
- Where a collision x, y is known, the realization can be written as $\langle \lambda r : \langle x, y\rangle, a\rangle$ (a having no algorithmic content)
- Where no collision is known, essentially the pigeonhole principle is realized

Hash collision as a positive formula

- $\exists A : P[A(r) = \langle x, y\rangle \wedge x \neq y \wedge h(x) = h(y)] > \varepsilon$ (r source of randomness)
- or: $\exists A : \neg(P[A(r) = \langle x, y\rangle \wedge x \neq y \wedge h(x) = h(y)] \leq \varepsilon)$
- A_{pre} is the algorithm extracted from the realization
- Where a collision x, y is known, the realization can be written as $\langle \lambda r : \langle x, y\rangle, a\rangle$ (a having no algorithmic content)
- Where no collision is known, essentially the pigeonhole principle is realized
- Proof possible in constructive mathematics, but leads to A_{pre} having a „long“ run time

Hash collision as a positive formula

- $\exists A : P[A(r) = \langle x, y \rangle \wedge x \neq y \wedge h(x) = h(y)] > \varepsilon$ (r source of randomness)
- or: $\exists A : \neg(P[A(r) = \langle x, y \rangle \wedge x \neq y \wedge h(x) = h(y)] \leq \varepsilon)$
- A_{pre} is the algorithm extracted from the realization
- Where a collision x, y is known, the realization can be written as $\langle \lambda r : \langle x, y \rangle, a \rangle$ (a having no algorithmic content)
- Where no collision is known, essentially the pigeonhole principle is realized
- Proof possible in constructive mathematics, but leads to A_{pre} having a „long" run time
- Or: $\langle a, b \rangle$, a being an „actual" attack algorithm

Pigeonhole principle, revisited

- Remember the math guy?

Pigeonhole principle, revisited

- Remember the math guy?
- Constructively, $\mathrm{card}(D) > \mathrm{card}(R)$ just proved that $\neg\forall xy : \neg(x \neq y \wedge h(x) = h(y))$

Pigeonhole principle, revisited

- Remember the math guy?
- Constructively, $\mathrm{card}(D) > \mathrm{card}(R)$ just proved that $\neg\forall xy : \neg(x \neq y \wedge h(x) = h(y))$
- Constructively, $\exists xy : x \neq y \wedge h(x) = h(y)$ cannot be derived just from this

Pigeonhole principle, revisited

- Remember the math guy?
- Constructively, $\text{card}(D) > \text{card}(R)$ just proved that $\neg\forall xy : \neg(x \neq y \wedge h(x) = h(y))$
- Constructively, $\exists xy : x \neq y \wedge h(x) = h(y)$ cannot be derived just from this
- This requires induction, thus leads to additional complexity

Complexity of precomputation

- $\exists A : P[A(r) = \langle x, y\rangle \wedge x \neq y \wedge h(x) = h(y)] > \varepsilon$

Complexity of precomputation

- $\exists A : P[A(r) = \langle x, y \rangle \wedge x \neq y \wedge h(x) = h(y)] > \varepsilon$
- requires: pigeonhole principle

Complexity of precomputation

- $\exists A : P[A(r) = \langle x, y \rangle \wedge x \neq y \wedge h(x) = h(y)] > \varepsilon$
- requires: pigeonhole principle
- requires: $\forall fxy : (\exists z : z < y \wedge f(z) = x) \vee \neg(\exists z : z < y \wedge f(z) = x)$

Summary

- Proof in constructive logic...

Summary

- Proof in constructive logic...
- ...leads to algorithm from the realization

Summary

- Proof in constructive logic...
- ...leads to algorithm from the realization
- The algorithm can be analyzed for its costs

Summary

- Proof in constructive logic...
- ...leads to algorithm from the realization
- The algorithm can be analyzed for its costs
- We cannot disprove that the collision exists (and shouldn't be able to)

Summary

- Proof in constructive logic...
- ...leads to algorithm from the realization
- The algorithm can be analyzed for its costs
- We cannot disprove that the collision exists (and shouldn't be able to)
- We *can* put a cost on its logical derivation

Formalizing collision resistance

- In the algorithm extracted from the realization, precomputation can only be explicit

Formalizing collision resistance

- In the algorithm extracted from the realization, precomputation can only be explicit
- Cost of the attack: $TIME(A_{pre}) + TIME(A)$

Formalizing collision resistance

- In the algorithm extracted from the realization, precomputation can only be explicit
- Cost of the attack: $TIME(A_{pre}) + TIME(A)$
- Problem: Algorithm A_{pre} only in lambda calculus for now - other models might be easier to examine

Formalizing collision resistance

- In the algorithm extracted from the realization, precomputation can only be explicit
- Cost of the attack: $TIME(A_{pre}) + TIME(A)$
- Problem: Algorithm A_{pre} only in lambda calculus for now - other models might be easier to examine
- Problem: possibly necessary to constructively prove theorems again that were already classically proved

Formalizing collision resistance

- In the algorithm extracted from the realization, precomputation can only be explicit
- Cost of the attack: $TIME(A_{pre}) + TIME(A)$
- Problem: Algorithm A_{pre} only in lambda calculus for now - other models might be easier to examine
- Problem: possibly necessary to constructively prove theorems again that were already classically proved
- Problem: checking costs in two tiers

Formalizing collision resistance

- In the algorithm extracted from the realization, precomputation can only be explicit
- Cost of the attack: $TIME(A_{pre}) + TIME(A)$
- Problem: Algorithm A_{pre} only in lambda calculus for now - other models might be easier to examine
- Problem: possibly necessary to constructively prove theorems again that were already classically proved
- Problem: checking costs in two tiers
- What happens to security reductions?

Thank you for your attention.

dreiwert@irc.hackint.org

Daniel J. Bernstein and Tanja Lange.
Non-uniform cracks in the concrete: the power of free precomputation

Ivan Dåmgard.
Collision free hash functions and public key signature schemes

Phillip Rogaway.
Formalizing Human Ignorance: Collision-Resistant Hashing without the Keys

Xiaoyun Wang and Hongbo Yu.
How to Break MD5 and Other Hash Functions