One-time trapdoor one-way functions

Julien Cathalo[1] and **Christophe Petit**[2]

[1] Smals [2] UCL Crypto Group

Trapdoors in cryptography

- One-way function (OWF)
  - Easy to compute but hard to invert

$$\begin{tikzcd} x \arrow[r, bend left, "\text{easy}"] & f(x) \arrow[l, bend left, dotted, "\text{hard}"] \end{tikzcd}$$

- Trapdoor one-way function (TOWF)
  - Can be inverted with the help of a *trapdoor*
  - Useful for public key cryptography :
    public key encryption, digital signatures,...

Trapdoors in cryptography

- One-way function (OWF)
  - Easy to compute but hard to invert

$$\begin{tikzcd} x \arrow[rr, bend left, "\text{easy}"] & & f(x) \arrow[ll, dotted, bend left, "\text{hard}"] \end{tikzcd}$$

- Trapdoor one-way function (TOWF)
  - Can be inverted with the help of a *trapdoor*
  - Useful for public key cryptography :
    public key encryption, digital signatures,...

- In general, a trapdoor gives some power to its holder

RSA and Rabin TOWF

- Oldest and most famous TOWF [RSA78,R79]
  $f : \mathbb{Z}_n^* \to \mathbb{Z}_n^* : x \to x^e \bmod n$
  RSA and Rabin differ in the choice of e

RSA and Rabin TOWF

- Oldest and most famous TOWF [RSA78,R79]
  $f : \mathbb{Z}_n^* \to \mathbb{Z}_n^* : x \to x^e \bmod n$
  RSA and Rabin differ in the choice of e
- One-way if $n = pq$, p and q large primes, and $1 < e < n$
- Trapdoor : given p, q we can invert f

RSA and Rabin TOWF

- Oldest and most famous TOWF [RSA78,R79]
  $f : \mathbb{Z}_n^* \rightarrow \mathbb{Z}_n^* : x \rightarrow x^e \bmod n$
  RSA and Rabin differ in the choice of e
- One-way if $n = pq$, p and q large primes, and $1 < e < n$
- Trapdoor : given p, q we can invert f
- Can we use the trapdoor and keep it secret ?
  Does inverting f reveal p, q ?

Leaking trapdoors

- Suppose Bob has a trapdoor.
  Alice chooses x, sends $y = f(x)$ to Bob.
  Bob uses the trapdoor to compute x' such that $f(x') = y$. Bob sends x' to Alice.

$$\xrightarrow{f(x)}$$

$$\xleftarrow{x'}$$

Leaking trapdoors

- Suppose Bob has a trapdoor.
  Alice chooses x, sends $y = f(x)$ to Bob.
  Bob uses the trapdoor to compute x' such that $f(x') = y$. Bob sends x' to Alice.

$$\xrightarrow{f(x)}$$
$$\xleftarrow{x'}$$

- RSA :
  - $\gcd(e, \varphi(n)) = 1$
  - $f : x \to x^e \bmod n$ is bijective
  - $x' = x$ so x' does not leak anything

Leaking trapdoors

- Suppose Bob has a trapdoor.
  Alice chooses x, sends $y = f(x)$ to Bob.
  Bob uses the trapdoor to compute x' such that $f(x') = y$. Bob sends x' to Alice.

$$\xrightarrow{f(x)}$$
$$\xleftarrow{x'}$$

- RSA :
  - $\gcd(e, \varphi(n)) = 1$
  - $f : x \to x^e \bmod n$ is bijective
  - $x' = x$ so x' does not leak anything
- Rabin :
  - $f : x \to x^2 \bmod n$ is four to one
  - $x' = x\epsilon$ where $\epsilon^2 = 1 \bmod n$ (there are 4 values)
  - If $\epsilon \neq \pm 1$, the trapdoor is leaked

Limiting the trapdoor's power

- Typically leaking trapdoors are undesirable... but what about *positive* applications ?

Limiting the trapdoor's power

- Typically leaking trapdoors are undesirable... but what about *positive* applications ?
- Useful to restrict the trapdoor's power
  - To ensure that it is used only once $\Rightarrow$ e-coins ?
  - To prove that it has been used $\Rightarrow$ right delegation system ?
  - To achieve some delayed fairness $\Rightarrow$ fair exchange ?

Outline

Introduction

One-time trapdoor one-way functions (OTTOWF)

Constructions

Fair exchange protocols

Conclusion

Outline

Introduction

One-time trapdoor one-way functions (OTTOWF)

Constructions

Fair exchange protocols

Conclusion

Definition

- A one-time trapdoor one-way function (OTTOWF) is given by 5 algorithms
  - **Setup** : generates $\langle k, t \rangle$ where k is a *key* (parameter) and t is the *trapdoor*
  - **Eval** : upon input of $\langle k, m \rangle$ where m is some *message*, outputs a *hash value* h
  - **Verify** : upon input of $\langle k, m, h \rangle$, outputs either 0 or 1

Definition

- A one-time trapdoor one-way function (OTTOWF) is given by 5 algorithms
  - **Setup** : generates $\langle k, t \rangle$ where k is a *key* (parameter) and t is the *trapdoor*
  - **Eval** : upon input of $\langle k, m \rangle$ where m is some *message*, outputs a *hash value h*
  - **Verify** : upon input of $\langle k, m, h \rangle$, outputs either 0 or 1
  - **Preimage** : upon input of $\langle k, h, t \rangle$, outputs a message m

Definition

- A one-time trapdoor one-way function (OTTOWF) is given by 5 algorithms
  - **Setup** : generates $\langle k, t \rangle$ where k is a *key* (parameter) and t is the *trapdoor*
  - **Eval** : upon input of $\langle k, m \rangle$ where m is some *message*, outputs a *hash value* h
  - **Verify** : upon input of $\langle k, m, h \rangle$, outputs either 0 or 1
  - **Preimage** : upon input of $\langle k, h, t \rangle$, outputs a message m
  - **TrapExtr** : upon input of $\langle k, m, m' \rangle$ where m, m' are two messages, outputs either $\perp$ or a trapdoor t

Definition

- These algorithms satisfy the following properties
  - **Correctness : Verify**(**Eval**) $= 1$

Definition

- These algorithms satisfy the following properties
  - **Correctness : Verify**(**Eval**) $= 1$
  - **Onewayness :** without the trapdoor, impossible to "invert" in the sense of satisfying **Verify**

Definition

- These algorithms satisfy the following properties
  - **Correctness : Verify**(**Eval**) $= 1$
  - **Onewayness :** without the trapdoor, impossible to "invert" in the sense of satisfying **Verify**
  - **Trapdoor :** with the trapdoor, possible to "invert" in the sense of satisfying **Verify**

Definition

- These algorithms satisfy the following properties
  - **Correctness : Verify**(**Eval**) $= 1$
  - **Onewayness :** without the trapdoor, impossible to "invert" in the sense of satisfying **Verify**
  - **Trapdoor :** with the trapdoor, possible to "invert" in the sense of satisfying **Verify**
  - **Fairness : TrapExtr** recovers the trapdoor if it gets two messages with the same hash value, one of them computed with the trapdoor

Definition remarks

- The trapdoor cannot be recovered with the key only

Definition remarks

- The trapdoor cannot be recovered with the key only
- An OTTOWF is not necessarily a TOWF
  - **Eval** can be probabilistic
  - **Correctness** does not require **Eval**(**Preimage**(h)) $= h$, but only that the **Preimage** algorithm finds a value that satisfies the **Verify** algorithm

Definition remarks

- The trapdoor cannot be recovered with the key only
- An OTTOWF is not necessarily a TOWF
  - **Eval** can be probabilistic
  - **Correctness** does not require **Eval**(**Preimage**(h)) $= h$, but only that the **Preimage** algorithm finds a value that satisfies the **Verify** algorithm
- An OTTOWF is a TOWF if **Eval** is deterministic and **Verify** just recomputes it

Outline

Introduction

One-time trapdoor one-way functions (OTTOWF)

Constructions

Fair exchange protocols

Conclusion

3 OTTOWF constructions

- 3 constructions
  - Tweak of Rabin's TOWF
  - Tweak of Paillier's TOWP
  - Based on generic OWF
- Various assumptions
  - Factoring assumption for $n = pq$
  - Factoring assumption for $n = p^2q$
  - OWF
- Different flavors of our definition

OTTOWF based on Rabin

- Tentative construction :
  - **Setup** : $t = \langle p, q \rangle$ and $k = n := pq$
  - **Eval** : given $\langle n, m \rangle$, returns $h = m^2 \bmod n$
  - **Verify** : given $\langle n, m, h \rangle$, outputs 1 iff $h = \textbf{Eval}(n, m)$
  - **Preimage** : given $\langle n, h, \langle p, q \rangle \rangle$, uses CRT to compute m' such that $m'^2 = h \bmod n$

OTTOWF based on Rabin

- Tentative construction :
  - **Setup** : $t = \langle p, q \rangle$ and $k = n := pq$
  - **Eval** : given $\langle n, m \rangle$, returns $h = m^2 \bmod n$
  - **Verify** : given $\langle n, m, h \rangle$, outputs 1 iff $h = \mathbf{Eval}(n, m)$
  - **Preimage** : given $\langle n, h, \langle p, q \rangle \rangle$, uses CRT to compute m' such that $m'^2 = h \bmod n$
  - **TrapExtr** : given $\langle k, m, m' \rangle$, computes $p' = \gcd(|m - m'|, n)$. Returns $(p', n/p')$

OTTOWF based on Rabin

- Tentative construction :
  - **Setup** : $t = \langle p, q\rangle$ and $k = n := pq$
  - **Eval** : given $\langle n, m\rangle$, returns $h = m^2 \bmod n$
  - **Verify** : given $\langle n, m, h\rangle$, outputs 1 iff $h = \mathbf{Eval}(n, m)$
  - **Preimage** : given $\langle n, h, \langle p, q\rangle\rangle$, uses CRT to compute m' such that $m'^2 = h \bmod n$
  - **TrapExtr** : given $\langle k, m, m'\rangle$, computes $p' = \gcd(|m - m'|, n)$. Returns $(p', n/p')$

- Fairness only satisfied with probability $1/2$
  - Trapdoor recoverd iff $m \neq \pm m'$
  - If $m = m'$ then $p' = n$
  - If $m = -m'$ then $p' = 1$

OTTOWF based on Rabin

- Improve fairness probability
  - Choose $m \in \mathbb{Z}_n^N$
  - **Eval** : given n and $m = (m_1, ..., m_N)$, returns $h = (m_1^2 \bmod n, ..., m_N^2 \bmod n)$

OTTOWF based on Rabin

- Improve fairness probability
  - Choose $m \in \mathbb{Z}_n^N$
  - **Eval** : given n and $m = (m_1, ..., m_N)$, returns $h = (m_1^2 \bmod n, ..., m_N^2 \bmod n)$
- Resulting OTTOWF
  - Fairness up to $1 - 2^{-N}$
  - Messages and hash values are quite long
  - Not surjective

OTTOWF based on Paillier

- Paillier [P99]
  - $f(m_1, m_2) = g^{m_1} m_2^n \bmod n$ with $n = p^2 q$ and $p \approx q$
  - $m_1 \approx p$ and $m_2 \approx pq \Rightarrow f \approx$ bijective
  - f can be inverted and inverse is unique

OTTOWF based on Paillier

- Paillier [P99]
  - $f(m_1, m_2) = g^{m_1} m_2^n \bmod n$ with $n = p^2 q$ and $p \approx q$
  - $m_1 \approx p$ and $m_2 \approx pq \Rightarrow f \approx$ bijective
  - f can be inverted and inverse is unique
- Idea : **make Paillier non injective**
  - $m_1 \approx p$ and $m_2 \approx p^2 q \Rightarrow f$ is many to one
  - f can be inverted but inverse not unique

OTTOWF based on Paillier

- Paillier [P99]
  - $f(m_1, m_2) = g^{m_1} m_2^n \bmod n$ with $n = p^2 q$ and $p \approx q$
  - $m_1 \approx p$ and $m_2 \approx pq \Rightarrow f \approx$ bijective
  - f can be inverted and inverse is unique
- Idea : **make Paillier non injective**
  - $m_1 \approx p$ and $m_2 \approx p^2 q \Rightarrow f$ is many to one
  - f can be inverted but inverse not unique
- **Two distinct inverses leak trapdoor**
  $f(m_1, m_2) = f(m_1', m_2') \Rightarrow m_1 = m_1' \bmod p$ and
  $f(m_1, m_2) = f(m_1, m_2') \Leftrightarrow m_2 = m_2' \bmod pq$

OTTOWF based on OWF

OTTOWF based on OWF

- **Setup**
  - Generate a OWF f
  - Trapdoor t is random domain element
  - Key $k = \langle f, \beta \rangle$ where $\beta := f(t)$
- **Eval** : given m returns $h = f(m)$

OTTOWF based on OWF

- **Setup**
  - Generate a OWF f
  - Trapdoor t is random domain element
  - Key $k = \langle f, \beta \rangle$ where $\beta := f(t)$
- **Eval** : given m returns $h = f(m)$
- **Verify** : given $\langle k, m, h \rangle$
  returns 1 if either $f(m) = h$ or $f(m) = \beta$

OTTOWF based on OWF

- **Setup**
  - Generate a OWF f
  - Trapdoor t is random domain element
  - Key $k = \langle f, \beta \rangle$ where $\beta := f(t)$
- **Eval** : given m returns $h = f(m)$
- **Verify** : given $\langle k, m, h \rangle$
  returns 1 if either $f(m) = h$ or $f(m) = \beta$
- **Preimage** : returns t

OTTOWF based on OWF

- **Setup**
  - Generate a OWF f
  - Trapdoor t is random domain element
  - Key $k = \langle f, \beta \rangle$ where $\beta := f(t)$
- **Eval** : given m returns $h = f(m)$
- **Verify** : given $\langle k, m, h \rangle$
  returns 1 if either $f(m) = h$ or $f(m) = \beta$
- **Preimage** : returns t
- **TrapExtr** : given $\langle t, m \rangle$ returns t

OTTOWF based on OWF

- **Onewayness** : to satisfy **Verify** we must invert f either on h or on β
- **Trapdoor** and **fairness** are clear
- Not a TOWF !

Outline

Introduction

One-time trapdoor one-way functions (OTTOWF)

Constructions

Fair exchange protocols

Conclusion

Signature scheme

Signature scheme

- **Setup** : create a pair (private key, public key) $\langle SK, PK \rangle$
- **Sign** : given a message m and a private key SK, returns a signature σ
- **Ver** : given a message, a public key, checks that the signature is valid for corresponding private key

Signature scheme

- **Setup** : create a pair (private key, public key) $\langle SK, PK\rangle$
- **Sign** : given a message m and a private key SK, returns a signature σ
- **Ver** : given a message, a public key, checks that the signature is valid for corresponding private key
- **Existential unforgeability against adaptive adversaries** : impossible to create a valid signature without the private key, even after seeing many valid signatures

The fair exchange problem

- Two parties want to exchange their own signatures
  Each party should get the other one's signature
  if and only if he has sent his own signature

The fair exchange problem

- Two parties want to exchange their own signatures
  Each party should get the other one's signature
  if and only if he has sent his own signature
- Issue : prevent abortions
- Typical protocols have four rounds
  - Exchange of *partial* signatures
  - Exchange of *full* signatures

The fair exchange problem

- Two parties want to exchange their own signatures
  Each party should get the other one's signature
  if and only if he has sent his own signature
- Issue : prevent abortions
- Typical protocols have four rounds
  - Exchange of *partial* signatures
  - Exchange of *full* signatures
- Semi-trusted third party (STTP) may help
  - Can convert partial signatures into full signatures
  - *Optimistic* if it only works in case of conflict

Verifiably committed signatures (VCS)

- State of the art with STTP [DR03]

Verifiably committed signatures (VCS)

- State of the art with STTP [DR03]
  - Exchange of partial signatures then full ones
  - STTP converts partial signatures into full ones (if one party aborts)
  - Full signatures $\approx$ converted partial signatures

Verifiably committed signatures (VCS)

- State of the art with STTP [DR03]
  - Exchange of partial signatures then full ones
  - STTP converts partial signatures into full ones (if one party aborts)
  - Full signatures $\approx$ converted partial signatures
- Fairness relies on STTP
  - OK if STTP honest
  - KO if STTP colludes with one of the parties

Concurrent signatures

- State of the art without STTP [CKP04]

Concurrent signatures

- State of the art without STTP [CKP04]
  - Partial *ambiguous* signatures are exchanged
  - Ambiguity removed when Initiator releases a *keystone*
    Full signature = partial signature + keystone

Concurrent signatures

- State of the art without STTP [CKP04]
  - Partial *ambiguous* signatures are exchanged
  - Ambiguity removed when Initiator releases a *keystone*
    Full signature = partial signature + keystone
- No STTP but *delayed* fairness :
  fairness iff the responder sees the keystone

Concurrent signatures

- State of the art without STTP [CKP04]
  - Partial *ambiguous* signatures are exchanged
  - Ambiguity removed when Initiator releases a *keystone*
    Full signature = partial signature + keystone
- No STTP but *delayed* fairness :
  fairness iff the responder sees the keystone
- Even the willingness to sign is hidden in partial signatures
- Partial signatures are non committing for the initiator

Fair exchange with OTTOWF

- General idea :
  - Take a randomness r, let $h = OTTOWF(r)$
  - Partial signature = signature on $\langle text, h \rangle$
    Full signature = $\langle$ partial signature, $r \rangle$

Fair exchange with OTTOWF

- General idea :
  - Take a randomness r, let $h = OTTOWF(r)$
  - Partial signature = signature on $\langle text, h \rangle$
    Full signature = $\langle$ partial signature, $r \rangle$
- Possible to check partial and full signatures

Fair exchange with OTTOWF

- General idea :
  - Take a randomness r, let $h = OTTOWF(r)$
  - Partial signature = signature on $\langle text, h \rangle$
    Full signature = $\langle$ partial signature, $r \rangle$
- Possible to check partial and full signatures
- **Conversion** partial signatures into full ones
  - Hard without the trapdoor
  - Easy with the trapdoor

Fair exchange with OTTOWF

- General idea :
  - Take a randomness r, let $h = OTTOWF(r)$
  - Partial signature = signature on $\langle text, h \rangle$
    Full signature = $\langle$ partial signature, $r \rangle$
- Possible to check partial and full signatures
- **Conversion** partial signatures into full ones
  - Hard without the trapdoor
  - Easy with the trapdoor
- **If used once, the trapdoor becomes public**
  If *one* partial signature converted, then *any* partial signature can be converted (fairness !)

Protocol 1 (no STTP)

- The initiator holds the trapdoor

1. (I) Generates OTTOWF ; keeps the trapdoor, sends the key.
   Generates r_I and sends $\sigma(text||OTTOWF(r_I))$
2. (R) Generates r_R and sends $\sigma(text||OTTOWF(r_R))$
3. (I) Sends r_I
4. (R) Sends r_R

Protocol 1 (no STTP)

- The initiator holds the trapdoor

1. (I) Generates OTTOWF ; keeps the trapdoor, sends the key. Generates r_I and sends $\sigma(text||OTTOWF(r_I))$
2. (R) Generates r_R and sends $\sigma(text||OTTOWF(r_R))$
3. (I) Sends r_I
4. (R) Sends r_R

- Fairness :
  - I or R aborts before Step 2 : none has full signature

Protocol 1 (no STTP)

- The initiator holds the trapdoor
  1. (I) Generates OTTOWF ; keeps the trapdoor, sends the key. Generates r_I and sends $\sigma(text||OTTOWF(r_I))$
  2. (R) Generates r_R and sends $\sigma(text||OTTOWF(r_R))$
  3. (I) Sends r_I
  4. (R) Sends r_R
- Fairness :
  - I or R aborts before Step 2 : none has full signature
  - R aborts after Step 3 : I uses his trapdoor

Protocol 1 (no STTP)

- The initiator holds the trapdoor

1. (I) Generates OTTOWF ; keeps the trapdoor, sends the key. Generates r_I and sends $\sigma(text||OTTOWF(r_I))$
2. (R) Generates r_R and sends $\sigma(text||OTTOWF(r_R))$
3. (I) Sends r_I
4. (R) Sends r_R

- Fairness :
  - I or R aborts before Step 2 : none has full signature
  - R aborts after Step 3 : I uses his trapdoor
  - I aborts after Step 2 and uses its trapdoor : the trapdoor is leaked, R can use it as well

Comparison with concurrent signatures

- *Delayed* fairness
  - If I uses his trapdoor to convert R's partial signature, then the resulting full signature leaks the trapdoor.
    R will therefore be able to convert I's partial signature as well, but *only after seeing this full signature*
  - Fairness also delayed in concurrent signatures

Comparison with concurrent signatures

- *Delayed* fairness
  - If I uses his trapdoor to convert R's partial signature, then the resulting full signature leaks the trapdoor. R will therefore be able to convert I's partial signature as well, but *only after seeing this full signature*
  - Fairness also delayed in concurrent signatures
- *Committing* partial signatures
  - Not true for concurrent signatures because of ambiguity

Protocol 2 (with an STTP)

- STTP has the trapdoor

  STTP generates an OTTOWF, keeps t, sends k

1. (I) Generates r_I and sends $\sigma(m||OTTOWF(r_I))$
2. (R) Generates r_R and sends $\sigma(m||OTTOWF(r_R))$
3. (I) Sends r_I
4. (R) Sends r_R

Protocol 2 (with an STTP)

- STTP has the trapdoor

  STTP generates an OTTOWF, keeps t, sends k

1. (I) Generates r_I and sends $\sigma(m||OTTOWF(r_I))$
2. (R) Generates r_R and sends $\sigma(m||OTTOWF(r_R))$
3. (I) Sends r_I
4. (R) Sends r_R

- If STTP honest :
  - If abortion after Step 2, STTP can convert partial signatures

Protocol 2 reduces the trust on the STTP

- If STTP and R collude :
  - R has the trapdoor.
    R aborts after Step 1 and converts I's partial signature
  - Fairness KO (same in VCS)

Protocol 2 reduces the trust on the STTP

- If STTP and R collude :
  - R has the trapdoor.
    R aborts after Step 1 and converts I's partial signature
  - Fairness KO (same in VCS)
- If STTP and I collude :
  - Amounts to Protocol 1 : I has the trapdoor
  - Still some delayed fairness (KO in VCS)

Protocol 2 reduces the trust on the STTP

- If STTP and R collude :
  - R has the trapdoor.
    R aborts after Step 1 and converts I's partial signature
  - Fairness KO (same in VCS)
- If STTP and I collude :
  - Amounts to Protocol 1 : I has the trapdoor
  - Still some delayed fairness (KO in VCS)

$\Rightarrow$ Compared to VCS, Protocol 2 reduces the trust that R needs to put on the STTP

Outline

Introduction

One-time trapdoor one-way functions (OTTOWF)

Constructions

Fair exchange protocols

Conclusion

Conclusion

- Leaking the trapdoor might be useful
- This paper :
  - OTTOWF definition
  - 3 constructions
  - Application to fair exchange
- Two new fair exchange protocols
  - Some advantages over previous protocols
- Other applications of OTTOWFs ?

Main references

- [RSA78] R Rivest, A Shamir, L Adleman, *A method for obtaining digital signatures and public-key cryptosystems*
- [R79] M Rabin, *Digitalized signatures and public key functions as intractable as factorization*
- [P99] P Paillier, *A trapdoor permutation equivalent to factoring*
- [CKP04] L Chen, C Kudla, K Paterson, *Concurrent signatures*
- [DR03] Y Dodis, L Reyzin, *Breaking and repairing optimistic fair exchange from PODC 2003*

Conclusion

- Leaking the trapdoor might be useful
- This paper :
  - OTTOWF definition
  - 3 constructions
  - Application to fair exchange
- Two new fair exchange protocols
  - Some advantages over previous protocols
- Other applications of OTTOWFs ?