# Onions Based on Universal Re–Encryption

Marcin Gomułkiewicz, Marek Klonowski and Mirek Kutyłowski

Wrocław University of Technology, Poland

Workshop on Information Security Applications (WISA) '2004, Jeju, Korea

# Communication systems

- messages can be kept secret
- reliable authentication
- how to hide that two parties are communicating??

# Need of anonymity in communication

- business to business communication
- consumer protection
- privacy protection
- economic and political security of a country

# Naive or local network solutions

- **all-to-all**: send the encrypted message to all participants, keep sending even if no message need to be sent communication overhead!!
- **token ring**: encoded messages go around the ring, only the legitimate recipient can understand it communication delay!!

# Major techniques for anonymous communication

- MIXes - David Chaum 1981
- DC-networks -David Chaum 1985
- Onions - Rackoff and Simon 1991,
  re-invented: Gülcü and Tsudik, 1996 (BABEL)
  Goldschlag, Reed, and Syverson, 1996 ( ONION ROUTING)

# Onions

If $A$ wants send a message $m$ to server $B$

- $A$ chooses at random $\lambda$ intermediate nodes $J_1, \ldots, J_\lambda$;

# Onions

If $A$ wants send a message $m$ to server $B$

- $A$ chooses at random $\lambda$ intermediate nodes $J_1, \ldots, J_\lambda$;
- $A$ creates an onion:
  $O :=$

$$\mathrm{Enc}_B(m)$$

# Onions

If $A$ wants send a message $m$ to server $B$

- $A$ chooses at random $\lambda$ intermediate nodes $J_1, \ldots, J_\lambda$;
- $A$ creates an onion:
  $O :=$

$$\mathrm{Enc}_{J_\lambda}(\mathrm{Enc}_B(m), B)$$

# Onions

If $A$ wants send a message $m$ to server $B$

- $A$ chooses at random $\lambda$ intermediate nodes $J_1, \ldots, J_\lambda$;
- $A$ creates an onion:
  $O :=$
  $$\mathrm{Enc}_{J_{\lambda-1}}(\mathrm{Enc}_{J_\lambda}(\mathrm{Enc}_B(m), B), J_\lambda)$$

# Onions

If $A$ wants send a message $m$ to server $B$

- $A$ chooses at random $\lambda$ intermediate nodes $J_1, \ldots, J_\lambda$;
- $A$ creates an onion:
  $O :=$
  $\mathrm{Enc}_{J_1}(\ldots(\mathrm{Enc}_{J_{\lambda-1}}(\mathrm{Enc}_{J_\lambda}(\mathrm{Enc}_B(m), B), J_\lambda), J_{\lambda-1})\ldots, J_2)$ .

# Processing an Onion

If $A$ wants send a message $m$ encrypted as $O$ to server $B$

- $A$ sends onion $O$ to $J_1$

# Processing an Onion

If $A$ wants send a message $m$ encrypted as $O$ to server $B$

- $A$ sends onion $O$ to $J_1$
- $J_1$ decrypts $O$ and obtains some $(O', J_2)$

# Processing an Onion

If $A$ wants send a message $m$ encrypted as $O$ to server $B$

- $A$ sends onion $O$ to $J_1$
- $J_1$ decrypts $O$ and obtains some $(O', J_2)$
- $J_1$ sends $O'$ to $J_2$

# Processing an Onion

If $A$ wants send a message $m$ encrypted as $O$ to server $B$

- $A$ sends onion $O$ to $J_1$
- $J_1$ decrypts $O$ and obtains some $(O', J_2)$
- $J_1$ sends $O'$ to $J_2$
- $J_2$ decrypts ..
- $J_2$ sends .. to $J_3$

# Processing an Onion

If $A$ wants send a message $m$ encrypted as $O$ to server $B$

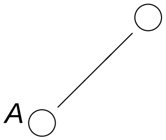
- $A$ sends onion $O$ to $J_1$
- $J_1$ decrypts $O$ and obtains some $(O', J_2)$
- $J_1$ sends $O'$ to $J_2$
- $J_2$ decrypts ..
- $J_2$ sends .. to $J_3$
- ...

# Route of an onion

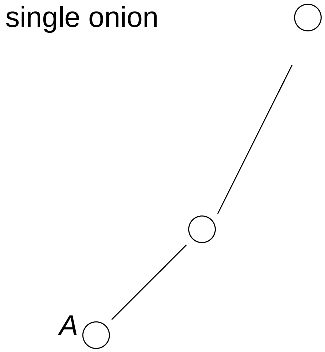
single onion

# Route of an onion

single onion

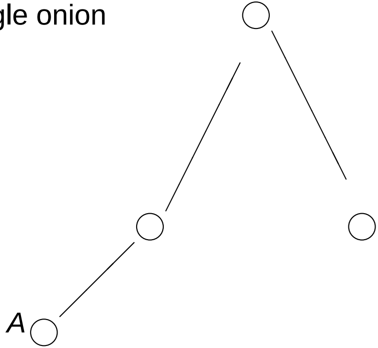


# Route of an onion



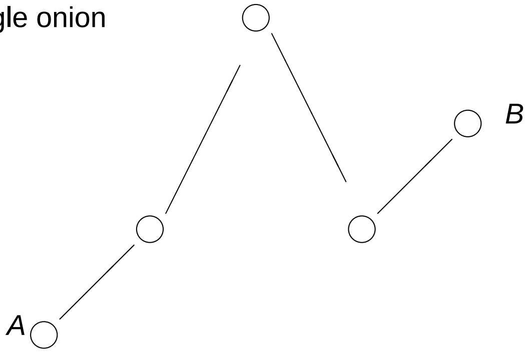
# Route of an onion

single onion



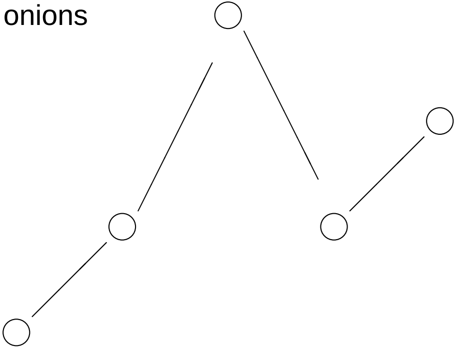
# Route of an onion

single onion



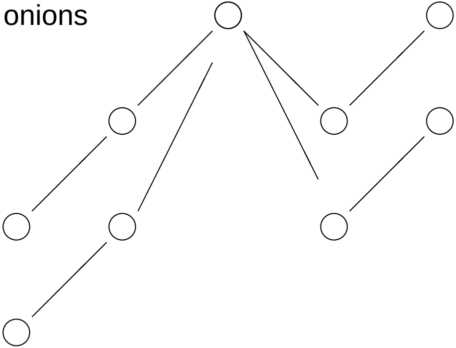
# Onions at work

many onions



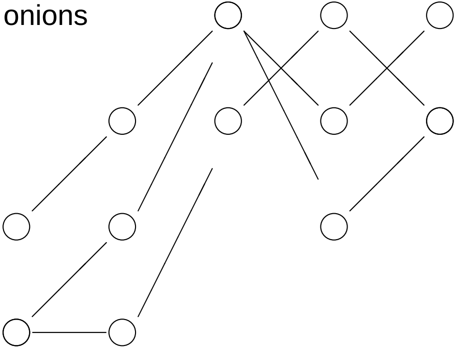
# Onions at work

many onions



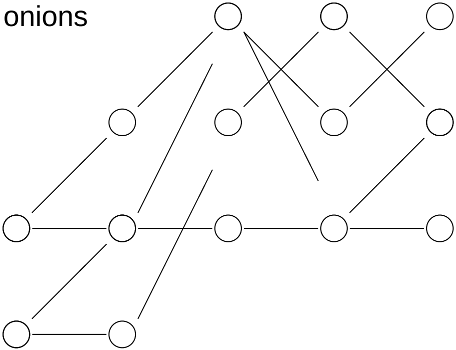
# Onions at work

many onions



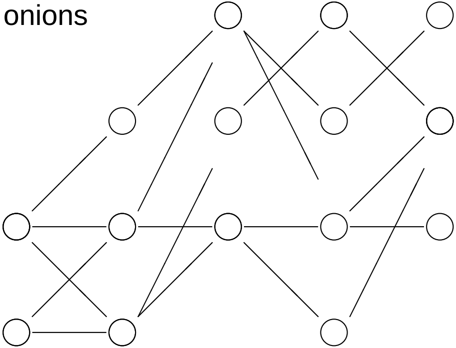
# Onions at work

many onions



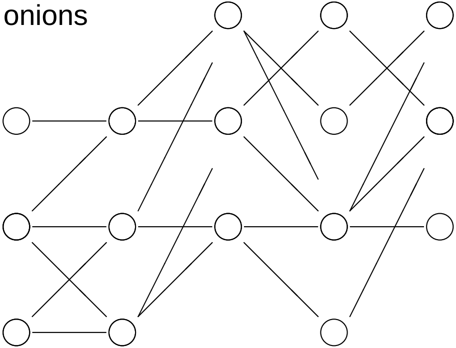
# Onions at work

many onions



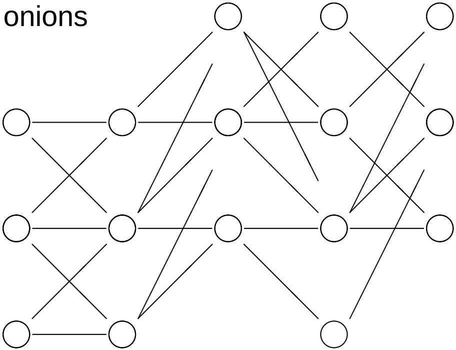
# Onions at work

many onions



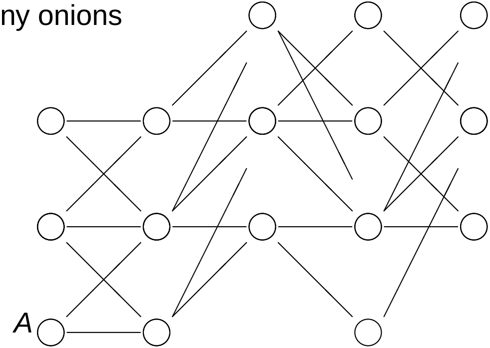
# Onions at work

many onions



# Onions at work

many onions




destination of the message starting at *A*?

# Viewpoint of an external observer

- no relationship can be derived between messages entering a node and leaving a node at the same time (probabilistic encryption has to be used)

# Viewpoint of an external observer

- no relationship can be derived between messages entering a node and leaving a node at the same time (probabilistic encryption has to be used)
- but: transmitting a message from a node to another node can be detected

# Traffic analysis

based on the traffic information and without breaking cryptographic functions

try to **determine any nontrivial relation between the senders and receivers**

# Adversaries

passive adversary :

- an adversary can monitor the whole traffic
- only a fraction of connections may be traced at each moment

# Adversaries

passive adversary :

- an adversary can monitor the whole traffic
- only a fraction of connections may be traced at each moment

active adversary : may influence the traffic

- non-adaptive (an attack cannot be adapted to the traffic observed)
- adaptive

# Security proofs for onions

**An adversary can monitor the whole traffic:**

- no security proof for the original protocol
- modified version of the protocol (routing in growing groups)
  Rackoff, Simon, FOCS'91, for $\lambda \approx \log^{11} n$,
  Czumaj, Kutyłowski, SODA'98, for $\lambda = O(\log^2 n)$

**Only a fraction of connections may be traced**

- Berman, Fiat, Ta-Shma, FC'2004, for $\lambda = O(\log^4 n)$
- Gomułkiewicz, Klonowski, Kutyłowski, ISC'2004, for $\lambda = \Theta(\log n)$

# Problems

- adversary analyzing system dynamics
  (emerging or disappearing connections)
- dynamic attacks (inserting and/or deleting messages)

# Dynamic attacks – repetitions

- an adversary re-sends the same onion

# Dynamic attacks – repetitions

- an adversary re-sends the same onion
- and observes where **duplicates** occur
  *path fully revealed without breaking cryptographic encoding*

# Countermeasures

- trace the traffic for duplicates
  *slow down, memory usage, intercepting log records easier than eavesdropping*

# Countermeasures

- trace the traffic for duplicates
  *slow down, memory usage, intercepting log records easier than eavesdropping*

- inserting “time to live” limits
  *limits disadvantages*

# Countermeasures -TOR

3rd Generation Onion Routing

- a path $A, J_1, J_2, \ldots, J_\lambda, B$ built up via messages:
  from $A$ to $J_1$,
  from $A$ to $J_2$,
  ...
  from $A$ to $J_\lambda$
- handshake mechanism for each connection

*high cost, attractive for establishing long-lasting connections*

# Universal re-encryption (URE)

- anybody can re-encrypt a ciphertext $C$
  so that without the private key one cannot find any relation between $C$ and the new ciphertext
- the public key is **not required**

# URE by Golle, Jakobsson, Juels, Syverson

- $p$ – prime such as for ElGamal encryption
- $x$ - private key
  $y = g^x \bmod p$ – public key

# URE by Golle, Jakobsson, Juels, Syverson

- $p$ – prime such as for ElGamal encryption
- $x$ - private key
  $y = g^x \mod p$ – public key
- ciphertext of $m$:

$$(a, b, c, d) = (m \cdot y^{k_1}, g^{k_1}, y^{k_2}, g^{k_2})$$

  for random $k_1, k_2$

# Re-encryption

Ciphertext : $(a, b, c, d) = (m \cdot y^{k_1}, g^{k_1}, y^{k_2}, g^{k_2})$

Re-encryption :

- random $r_1, r_2$
- $a := a \cdot c^{r_1}, \quad b := b \cdot d^{r_1}$
- $c := c^{r_2}, \quad d := d^{r_2}$

New ciphertext :

$$(a', b', c', d') = (m \cdot y^{k_1 + k_2 \cdot r_1}, g^{k_1 + k_2 \cdot r_1}, y^{k_2 \cdot r_2}, g^{k_2 \cdot r_2})$$

# URE-onions

- an URE-onion consists of $\lambda$ blocks
- a block = URE ciphertext
- encoded plaintexts:
  $J_2, J_3, \ldots, J_\lambda, m$
- advantage: each block can be re-encrypted while processing at a server
  **repetitions get undetectable!**
- no extra random content encoded

# URE-onions - partial decryption

**Goal: enforce processing along the path**

- $y_1, ..., y_\lambda$ = public keys of $J_1, \ldots, J_\lambda$
- ciphertext of $J_i$ encoded with the public key $y_1 \cdot y_2 \cdot \ldots \cdot y_{i-1}$:

$$(J_i \cdot (y_1 \cdot y_2 \cdot \ldots \cdot y_{i-1})^k, g^k, (y_1 \cdot y_2 \cdot \ldots \cdot y_{i-1})^{k'}, g^{k'})$$

# URE-onions - partial decryption

**Goal: enforce processing along the path**

- $y_1,...,y_\lambda$ = public keys of $J_1, \ldots, J_\lambda$
- ciphertext of $J_i$ encoded with the public key $y_1 \cdot y_2 \cdot \ldots \cdot y_{i-1}$:

$$(J_i \cdot (y_1 \cdot y_2 \cdot \ldots \cdot y_{i-1})^k, g^k, (y_1 \cdot y_2 \cdot \ldots \cdot y_{i-1})^{k'}, g^{k'})$$

- partial decryption of $(a, b, c, d)$ by $J_1$:

$$a := a / b^{x_1}, \quad c := c / d^{x_1}$$

# URE-onions - partial decryption

**Goal: enforce processing along the path**

- $y_1, ..., y_\lambda$ = public keys of $J_1, \ldots, J_\lambda$
- ciphertext of $J_i$ – with the public key $y_1 \cdot y_2 \cdot \ldots \cdot y_{i-1}$:

$$(J_i \cdot (\mathbf{y_1} \cdot \mathbf{y_2} \cdot \ldots \cdot y_{i-1})^k, g^k, (\mathbf{y_1} \cdot \mathbf{y_2} \cdot \ldots \cdot y_{i-1})^{k'}, g^{k'})$$

- partial decryption of $(a, b, c, d)$ by $J_1$:

$$a := a / b^{x_1}, \quad c := c / d^{x_1}$$

Result:

$$(J_i \cdot (\mathbf{y_2} \cdot \ldots \cdot y_{i-1})^k, g^k, (\mathbf{y_2} \cdot \ldots \cdot y_{i-1})^{k'}, g^{k'})$$

# Processing an onion

- partial decryption of all blocks
  next hop address $J_i$ or $m$ retrieved

# Processing an onion

- partial decryption of all blocks
  next hop address $J_i$ or $m$ retrieved
- re-encryption of all blocks

# Processing an onion

- partial decryption of all blocks
  next hop address $J_i$ or $m$ retrieved
- re-encryption of all blocks
- random permutation of all blocks

# Processing an onion

- partial decryption of all blocks
  next hop address $J_i$ or $m$ retrieved
- re-encryption of all blocks
- random permutation of all blocks
- delivery to $J_i$ or to the final destination

# Advantages

- the same onion sent twice is re-encrypted in a different way -repetitive attack does not work

# Advantages

- the same onion sent twice is re-encrypted in a different way -repetitive attack does not work
- partial decryption enforces that an URE-onion has to be decrypted by appropriate servers in a certain order
- it prohibits adding additional layers

# Disadvantages

- size
- computational effort
- how to combine URE with symmetric encryption in a secure and efficient way?

# Multiplicative attack

- $a := a \cdot u$
  it converts a ciphertext of $z$ to a ciphertext of $z \cdot u$
- $\Rightarrow$ destroys an address or a message

# Multiplicative attack

- $a := a \cdot u$
  it converts a ciphertext of $z$ to a ciphertext of $z \cdot u$
- $\Rightarrow$ destroys an address or a message
- there is a straightforward investigation that detects a malicious server

# Re-direction attack

- let an URE-onion use a path $J_1, J_2, J_3, \ldots$
- let $J_1$ be corrupted,
  it knows $J_2$, but not $J_3$, even if $J_3$ is corrupted

# Re-direction attack

- let an URE-onion use a path $J_1$, $J_2$, $J_3$, ...
- let $J_1$ be corrupted,
  it knows $J_2$, but not $J_3$, even if $J_3$ is corrupted
- attack by $J_1$:
  1. remove the block with the (encrypted) address of $J_3$
  2. insert a block with the address of $J_z$, where $J_z$ is also corrupted

# Re-direction attack

- let an URE-onion use a path $J_1, J_2, J_3, \ldots$
- let $J_1$ be corrupted,
  it knows $J_2$, but not $J_3$, even if $J_3$ is corrupted
- attack by $J_1$:
  1. remove the block with the (encrypted) address of $J_3$
  2. insert a block with the address of $J_z$, where $J_z$ is also corrupted
  3. after a while $J_z$ obtains this URE-onion but re-coded

# Re-direction attack

- let an URE-onion use a path $J_1, J_2, J_3, \ldots$
- let $J_1$ be corrupted,
  it knows $J_2$, but not $J_3$, even if $J_3$ is corrupted
- attack by $J_1$:
  1. remove the block with the (encrypted) address of $J_3$
  2. insert a block with the address of $J_z$, where $J_z$ is also corrupted
  3. after a while $J_z$ obtains this URE-onion but re-coded
  4. decoding with the key of $J_z$ yields garbage
     $\Rightarrow J_z$ makes trial decryptions with all private keys of corrupted servers

# Re-direction attack

- let an URE-onion use a path $J_1$, $J_2$, $J_3$, ...
- let $J_1$ be corrupted,
  it knows $J_2$, but not $J_3$, even if $J_3$ is corrupted
- attack by $J_1$:
  1. remove the block with the (encrypted) address of $J_3$
  2. insert a block with the address of $J_z$, where $J_z$ is also corrupted
  3. after a while $J_z$ obtains this URE-onion but re-coded
  4. decoding with the key of $J_z$ yields garbage
     $\Rightarrow$ $J_z$ makes trial decryptions with all private keys of corrupted servers
  5. if $J_z$ obtains a valid address with the private key of $J_i$, then the original processing is resumed at $J_i$

# Re-direction attack

- a partial disclosure of a path becomes possible, despite of re-encryption

# Re-direction attack

- a partial disclosure of a path becomes possible, despite of re-encryption
- **but**: if the wrong block removed, then the next server obtains two addresses of the next hop
  – a straightforward investigation and proof of malicious behavior

# Further possibilities with URE-onions

- implementing onions in a layered communication architecture:
  - offline preparation of onions
  - delegating construction of the path to other communication servers
    (*adopting path length to traffic intensity*, ...)

# Further possibilities with URE-onions

- implementing onions in a layered communication architecture:
  - offline preparation of onions
  - delegating construction of the path to other communication servers
    (*adopting path length to traffic intensity*, ...)
- signing onions with re-encryption of signatures

Thanks for your attention!

special thanks to an anonymous reviewer