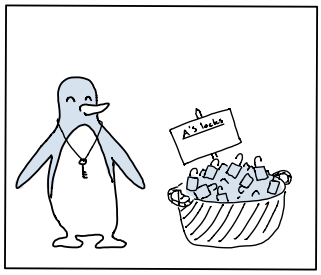
Public-Key Cryptography – the basic idea

Definitions

pke is a public-key encryption scheme if it satisfies:



Syntax:

$(pk, sk) \leftarrow \$\ pke.kgen(1^\lambda)$

or

$c \leftarrow \$\ pke.enc(pk, x)$

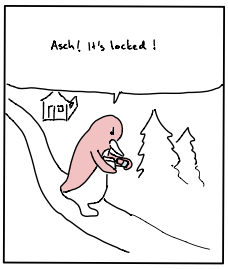
$y \leftarrow \$\ pke.dec(sk, c)$

poly-time in λ

Correctness:

$$Pr\{pke.dec(sk, c) = x\} = 1$$

Security:



$G^{ind\text{-}cpa^0}_{pke}$:

GETPK()
if $pk = \perp$
  $(pk, sk) \leftarrow \$\ pke.kgen(1^\lambda)$
return pk

ENC(x)
if $pk = \perp$
  $(pk, sk) \leftarrow \$\ pke.kgen(1^\lambda)$
$c \leftarrow \$\ pke.enc(pk, x)$
return c

$G^{ind\text{-}cpa^1}_{pke}$:

GETPK()
if $pk = \perp$
  $(pk, sk) \leftarrow \$\ pke.kgen(1^\lambda)$
return pk

ENC(x)
if $pk = \perp$
  $(pk, sk) \leftarrow \$\ pke.kgen(1^\lambda)$
$x' \leftarrow 0^{|x|}$
$c \leftarrow \$\ pke.enc(pk, x')$
return c

$$\forall \text{ PPT } \mathcal{A}: \left| Pr[1 = \mathcal{A} \rightarrow G^{ind\text{-}cpa^0}_{pke}] - Pr[1 = \mathcal{A} \rightarrow G^{ind\text{-}cpa^1}_{pke}] \right| = \text{negl. in } \lambda$$

Claim 1 IND-CPA secure PKE exists $\Rightarrow$ OWFs exist

proof: (sketch)

Suppose IND-CPA secure PKE pke exists

Now the function

$f(r \| r' \| x)$
$(pk, sk) \leftarrow pke.kgen(1^\lambda; r')$
$c \leftarrow pke.enc(pk, x; r)$
return $c \| pk$

is a OWF

why?

Assume for contr. that f is not OWF

i.e. $\exists\ \mathcal{A}$ s.t. $Pr[y' \leftarrow \mathcal{A}(f(y))] =$ non-negl.

Consider:

pke

$\mathcal{B}(1^\lambda)$
$pk \leftarrow \text{GETPK}()$
$x \leftarrow\$ \{0,1\}^\lambda$
$c \leftarrow\$ \text{ENC}(x)$
$r \| r' \| x' \leftarrow \mathcal{A}(c \| pk)$
if $pke.enc(pk, x'; r) = c$ AND $pke.kgen(1^\lambda; r') = (pk, sk)$
  if $x = x'$
    return 1
return 0

Claim 2 OWF exists $\Rightarrow$ IND-CPA secure PKE exists

We don't know whether this is true! :(

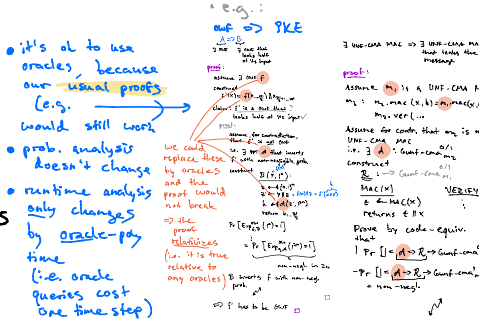
↳ we believe OWFs and PKE exist

... and, we know that proving claim 2 is hard

Claim 3 One cannot prove claim 2 in a black-box way.

What is "black-box way"?

- this is the way we usually prove things
- e.g.:



- it's ok to use oracles, because our usual proofs (e.g. would still work)
- prob. analysis doesn't change
- runtime analysis only changes by oracle-pay time (i.e. oracle queries cost one time step)

proof:

idea:

introduce oracles s.t. $\exists$ OWF but $\nexists$ PKE when "everyone" has access to the oracles

i.e. the OWF, PKE and the adversaries can make calls to the oracles

Which oracle rules out PKE?

- PSPACE-oracle! i.e. an oracle that can solve any problem solvable in polynomial (w.r.t. problem size) space

  why? - in poly space one can exhaustively find the secret key

problem: PSPACE oracle also rules out OWFs

↓ (every OWF can be inverted by exhaustively trying all inputs)

Which oracle would add OWFs to our PSPACE = P world — without adding PKE?

all problems solvable in poly time

- random function oracle!

  i.e. oracle $RO(x)$
  if $L[x] = \perp$
    $L[x] \leftarrow\$ \{0,1\}^\lambda$

return $L[x]$

Prove that RO is OWF

Claim: let $f(x) := RO(x)$.
Now f is OWF in the "oracle world", i.e.
$\forall \mathcal{A}$ ($\mathcal{A}$ can make queries to RO and PSPACE)
s.t. $\mathcal{A}$ makes at most $q(\lambda)$ (poly) queries to RO:

$$\Pr_{x \leftarrow \$\{0,1\}^\lambda}\left[\mathcal{A}^{RO}(f(x), 1^\lambda) \in f^{-1}(f(x))\right] \leq \text{negl. in } \lambda$$

proof:

WLOG assume $\mathcal{A}$ make exactly q queries to RO.
Denote the answers by $x_1, \dots, x_q$ and denote the output of $\mathcal{A}$ by x_{q+1}

Now $\Pr\left[\mathcal{A}^{RO}(f(x), 1^\lambda) \in f^{-1}(f(x))\right]$

$$\leq \sum_{i=1}^{q+1} \Pr\left[f(x_i) = f(x) \mid f(x_1), \dots, f(x_{i-1}) \neq f(x)\right]$$

if $x_i \neq x$ the prob is $\leq \frac{1}{2^\lambda - i + 1}$ (# of x s.t. $x \neq x_1, \dots, x_{i-1}$)

if $x_i = x$ the prob is $\leq \frac{1}{2^\lambda}$

$$\leq \sum_{i=1}^{q+1} \frac{1}{2^\lambda - i + 1} + \frac{1}{2^\lambda} \qquad (\leq q)$$

$$\leq \sum \frac{1}{2^\lambda - q} + \frac{1}{2^\lambda - q} \qquad \left(\leq \frac{1}{\frac{1}{2}2^\lambda}\right)$$

$$\leq \sum \frac{1}{\frac{1}{2}2^\lambda} + \frac{1}{\frac{1}{2}2^\lambda} = \sum \frac{2}{\frac{1}{2}2^\lambda} = \sum \frac{1}{\frac{1}{4}2^\lambda}$$

$$= (q+1) \cdot \frac{1}{\frac{1}{4}2^\lambda} = \underbrace{(q+1) \cdot 4}_{\text{poly}} \cdot \underbrace{\frac{1}{2^\lambda}}_{\text{negl.}}$$

$= \text{negl. in } \lambda$ □

Prove that we cannot get PKE from RO

PSPACE

High-level idea:

- $\mathcal{A}$ knows pk, c, and the program code of pke
- Suppose enc was making RO(v) for some v
  - ⇒ dec must also know RO(v)
  - ⇒ sk, c, or dec must contain v or RO(v) ($\mathcal{A}$ knows)
    - kgen must know v or RO(v)
    - ⇒ v or RO(v) is part of kgen or pk ($\mathcal{A}$ knows)

⇒ pke should not benefit from the RO calls!

$x \leftarrow dec^{RO'}(sk, c)$

Let's prove this rigorously:

denote $pke.enc(pk, x; r_e)$ (internal randomness of enc)
$pke.kgen(1^\lambda; r_k)$

$Q_e = \{(q,a) \mid$ enc makes query q to RO and $RO(q) = a\}$

$Q_{k,d} = \{(q,a) \mid$ kgen or dec makes query q to RO and $a = RO(q)\}$

$Q_{\mathcal{A}} =$ "the guess of $\mathcal{A}$ for $Q_e \cap Q_{k,d}$"

$\mathcal{A}^{RO, PSPACE}(1^\lambda)$

$x \leftarrow \$ \{0,1\}^\lambda$
$c \leftarrow \$ ENC(x)$
$pk \leftarrow \$ GETPK()$
$Q_{\mathcal{A}}, \bar{X} \leftarrow \phi$
for $i = 1, \dots, 2 \cdot |Q_{k,d}| + 1$ (max # of queries to RO)
  1) find $w' = (x', r_e, q'_1, a'_1, \dots, q'_{|Q_e|}, a'_{|Q_e|})$ s.t. w' is consistent with a run of $enc(pk, x'; r_e) = c$ and consistent $Q_{\mathcal{A}}$ (using PSPACE)
  2) for $q \in w'$: $a \leftarrow RO(q)$; add (q,a) to $Q_{\mathcal{A}}$
  3) add x' to $\bar{X}$
if x is the most common elem in $\bar{X}$ return 1 else return 0

1) $w' \cap Q_{k,d} = \phi$

$RO'(x)$
if $x \in w'$
  return corresp. $a \in w'$
else
  return $RO(x)$

x' is correct

2) $w' \cap Q_{k,d} \neq \phi$
" this

adds $q \in Q_{k,d}$ into Q_d

Why does d succeed?

Lemma for every i d adds a new query $q \in Q_{k,d}$ to Q_d or it adds the correct message (the message encrypted by ENC) to $\tilde{X}$.

□

this can happen at most $|Q_{k,d}|$ times

$\Rightarrow$ the majority of $\tilde{X}$ will be correct