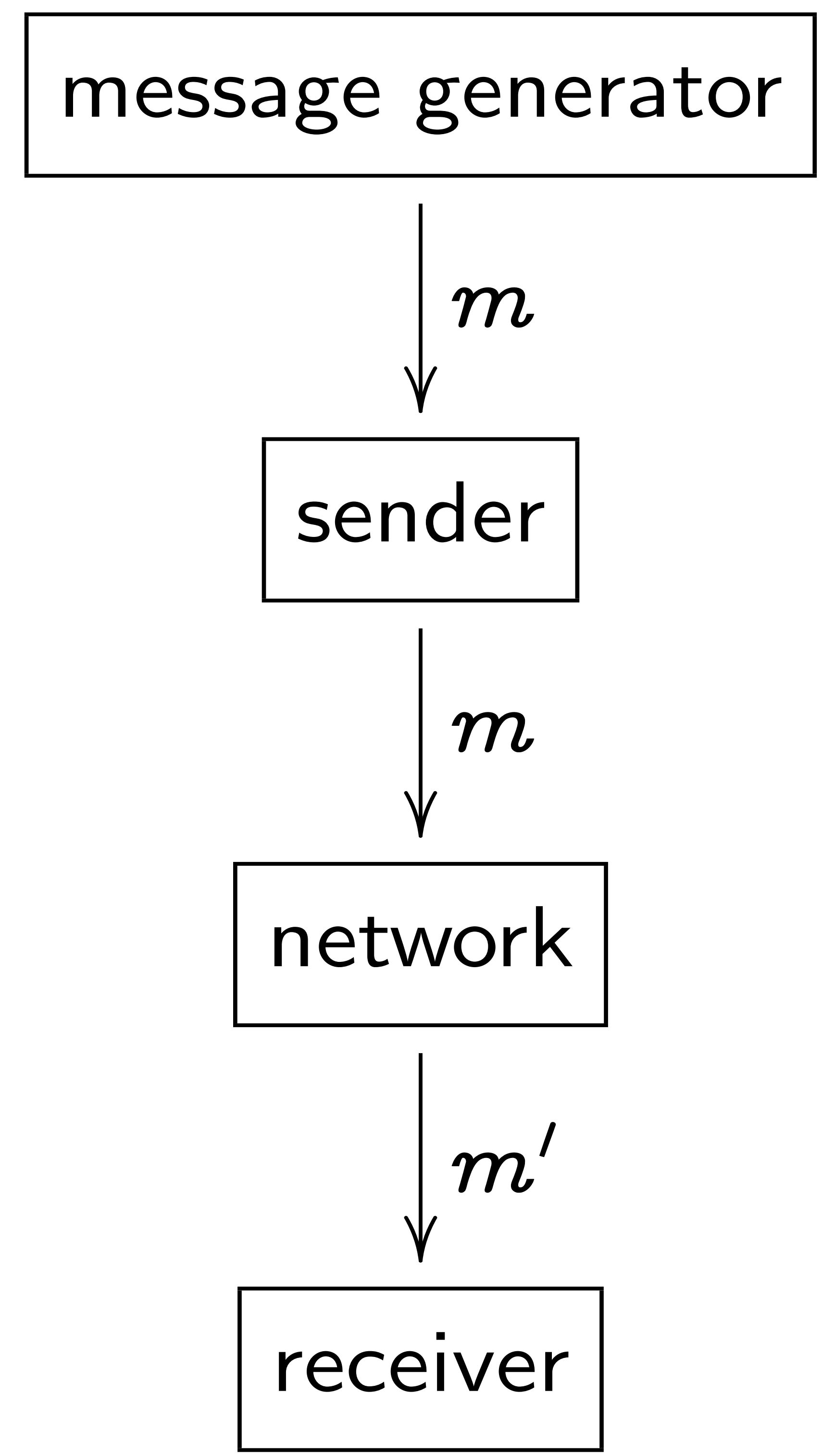
High-speed cryptographic functions

D. J. Bernstein

Typical Internet protocol:



nctions

Typical Internet protocol:

$$\begin{tikzcd}
\boxed{\text{message generator}} \arrow[d, "m"] \\
\boxed{\text{sender}} \arrow[d, "m"] \\
\boxed{\text{network}} \arrow[d, "m'"] \\
\boxed{\text{receiver}}
\end{tikzcd}$$

A **message gene**
creates a **messag**
a string of bytes.

Message generat
gives m to a **sen**

Sender gives m t

Network gives a
to a **receiver**.

Maybe $m' = m$;
Maybe network is
controlled by an
who changed m

Typical Internet protocol:

$$\begin{tikzcd}
\boxed{\text{message generator}} \arrow[d, "m"] \\
\boxed{\text{sender}} \arrow[d, "m"] \\
\boxed{\text{network}} \arrow[d, "m'"] \\
\boxed{\text{receiver}}
\end{tikzcd}$$

A **message generator** creates a **message** m, a string of bytes.

Message generator gives m to a **sender**.

Sender gives m to a **network**.

Network gives a message m' to a **receiver**.

Maybe $m' = m$; maybe not. Maybe network is controlled by an attacker who changed m into $m' \neq m$.

protocol:

tor

A **message generator** creates a **message** m, a string of bytes.

Message generator gives m to a **sender**.

Sender gives m to a **network**.

Network gives a message m' to a **receiver**.

Maybe $m' = m$; maybe not. Maybe network is controlled by an attacker who changed m into $m' \neq m$.

Protocol eliminat

message genera

$\downarrow m$

sender using r

$\downarrow m, a$

network

$\downarrow m', a$

receiver using r

A **message generator** creates a **message** m, a string of bytes.

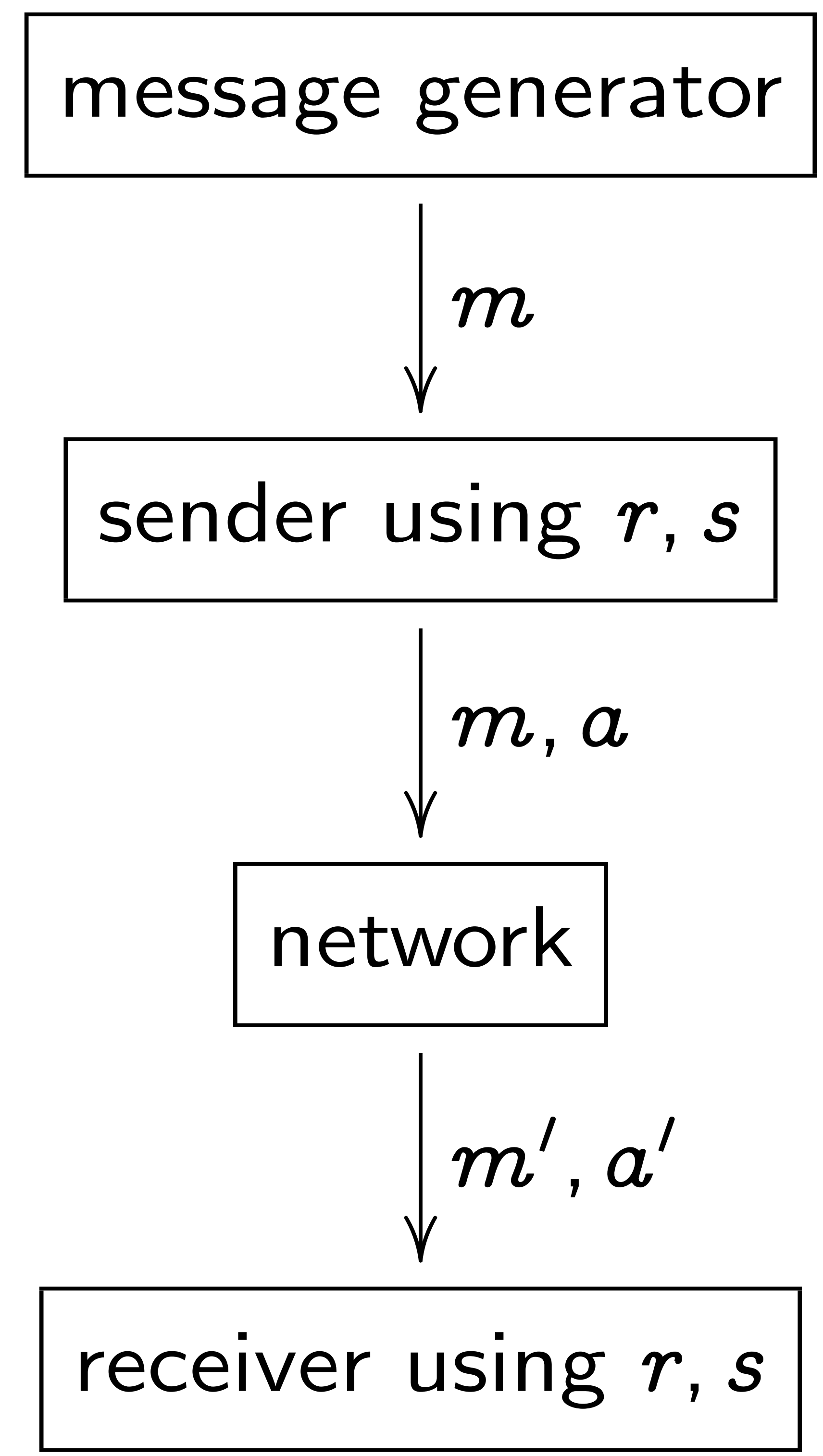
Message generator gives m to a **sender**.

Sender gives m to a **network**.

Network gives a message m' to a **receiver**.

Maybe $m' = m$; maybe not. Maybe network is controlled by an attacker who changed m into $m' \neq m$.

Protocol eliminating forgeries:



erator
ge m,

or
nder.

to a **network**.

message m'

maybe not.
s
attacker
into $m' \neq m$.

Protocol eliminating forgeries:

$$
\begin{tikzcd}
\boxed{\text{message generator}} \arrow[d, "m"] \\
\boxed{\text{sender using } r, s} \arrow[d, "{m, a}"] \\
\boxed{\text{network}} \arrow[d, "{m', a'}"] \\
\boxed{\text{receiver using } r, s}
\end{tikzcd}
$$

Fix a finite field
Typically $\#k \approx$

Sender, receiver
uniform random
Network's functi
is independent o

Sender encodes
as polynomial $\underline{m}$
Sender then com
authenticator a

Receiver discards
if $a' \neq \underline{m'}(r) +$

Protocol eliminating forgeries:

$$\begin{tikzcd}
\boxed{\text{message generator}} \arrow[d, "m"] \\
\boxed{\text{sender using } r, s} \arrow[d, "m, a"] \\
\boxed{\text{network}} \arrow[d, "m', a'"] \\
\boxed{\text{receiver using } r, s}
\end{tikzcd}$$

Fix a finite field k.
Typically $\#k \approx 2^{128}$.

Sender, receiver share a **secret**: uniform random $(r, s) \in k \times k$.
Network's function $m, a \mapsto m', a'$ is independent of (r, s).

Sender encodes message m as polynomial $\underline{m} \in xk[x]$.
Sender then computes **authenticator** $a = \underline{m}(r) + s$.

Receiver discards m', a' if $a' \neq \underline{m'}(r) + s$.

ting forgeries:

ntor

, s

$'$

r, s

Fix a finite field k.
Typically $\#k \approx 2^{128}$.

Sender, receiver share a **secret**: uniform random $(r, s) \in k \times k$.
Network's function $m, a \mapsto m', a'$ is independent of (r, s).

Sender encodes message m as polynomial $\underline{m} \in xk[x]$.
Sender then computes **authenticator** $a = \underline{m}(r) + s$.

Receiver discards m', a' if $a' \neq \underline{m'}(r) + s$.

If $m' \neq m$ then
Pr[receiver accep
$\leq$ max$\{$deg $\underline{m}$, d

e.g. Pr $\leq 2^{-98}$ i
and message deg

Proof: $\underline{m'} \neq \underline{m}$ i
so $\underline{m'} - a' \neq \underline{m}$
$\#k$ pairs $(r, s) \in$
satisfy $a = \underline{m}(r)$
$\leq$ max$\{$deg $\underline{m}$, d
also satisfy $a' =$
$\square$

Fix a finite field k.
Typically $\#k \approx 2^{128}$.

Sender, receiver share a **secret**:
uniform random $(r, s) \in k \times k$.
Network's function $m, a \mapsto m', a'$
is independent of (r, s).

Sender encodes message m
as polynomial $\underline{m} \in xk[x]$.
Sender then computes
authenticator $a = \underline{m}(r) + s$.

Receiver discards m', a'
if $a' \neq \underline{m'}(r) + s$.

If $m' \neq m$ then
$\Pr[\text{receiver accepts } m']$
$\leq \max\{\deg \underline{m}, \deg \underline{m'}\} / \#k$.

e.g. $\Pr \leq 2^{-98}$ if $\#k = 2^{128}$
and message degree $\leq 2^{30}$.

Proof: $\underline{m'} \neq \underline{m}$ in $xk[x]$
so $\underline{m'} - a' \neq \underline{m} - a$ in $k[x]$.
$\#k$ pairs $(r, s) \in k \times k$
satisfy $a = \underline{m}(r) + s$.
$\leq \max\{\deg \underline{m}, \deg \underline{m'}\}$ pairs
also satisfy $a' = \underline{m'}(r) + s$.
$\square$

k.

2^{128}.

share a **secret**:

$(r, s) \in k \times k$.

on $m, a \mapsto m', a'$

f (r, s).

message m

$\in xk[x]$.

putes

$= \underline{m}(r) + s$.

s m', a'

s.

If $m' \neq m$ then

Pr[receiver accepts m']

$\leq \max\{\deg \underline{m}, \deg \underline{m'}\}/\#k$.

e.g. Pr $\leq 2^{-98}$ if $\#k = 2^{128}$

and message degree $\leq 2^{30}$.

Proof: $\underline{m'} \neq \underline{m}$ in $xk[x]$

so $\underline{m'} - a' \neq \underline{m} - a$ in $k[x]$.

$\#k$ pairs $(r, s) \in k \times k$

satisfy $a = \underline{m}(r) + s$.

$\leq \max\{\deg \underline{m}, \deg \underline{m'}\}$ pairs

also satisfy $a' = \underline{m'}(r) + s$.

$\square$

Many messages,

message genera

$\downarrow m_1; n$

sender

$\downarrow 1, m_1$

network

$\downarrow n'_1, m$

receiver

If $m' \neq m$ then
Pr[receiver accepts m']
$\leq \max\{\deg \underline{m}, \deg \underline{m'}\}/\#k$.

e.g. $\Pr \leq 2^{-98}$ if $\#k = 2^{128}$
and message degree $\leq 2^{30}$.

Proof: $\underline{m'} \neq \underline{m}$ in $xk[x]$
so $\underline{m'} - a' \neq \underline{m} - a$ in $k[x]$.
$\#k$ pairs $(r, s) \in k \times k$
satisfy $a = \underline{m}(r) + s$.
$\leq \max\{\deg \underline{m}, \deg \underline{m'}\}$ pairs
also satisfy $a' = \underline{m'}(r) + s$.
$\square$

Many messages, unprotected:

$$\begin{tikzcd}
\boxed{\text{message generator}} \arrow[d, "m_1; m_2; \ldots"] \\
\boxed{\text{sender}} \arrow[d, "1, m_1; 2, m_2; \ldots"] \\
\boxed{\text{network}} \arrow[d, "n'_1, m'_1; n'_2, m'_2; \ldots"] \\
\boxed{\text{receiver}}
\end{tikzcd}$$

ots $m']$

eg $\underline{m'}\}/\#k$.

f $\#k = 2^{128}$

gree $\le 2^{30}$.

in $xk[x]$

$- a$ in $k[x]$.

$k \times k$

$+ s$.

eg $\underline{m'}\}$ pairs

$\underline{m'}(r) + s$.

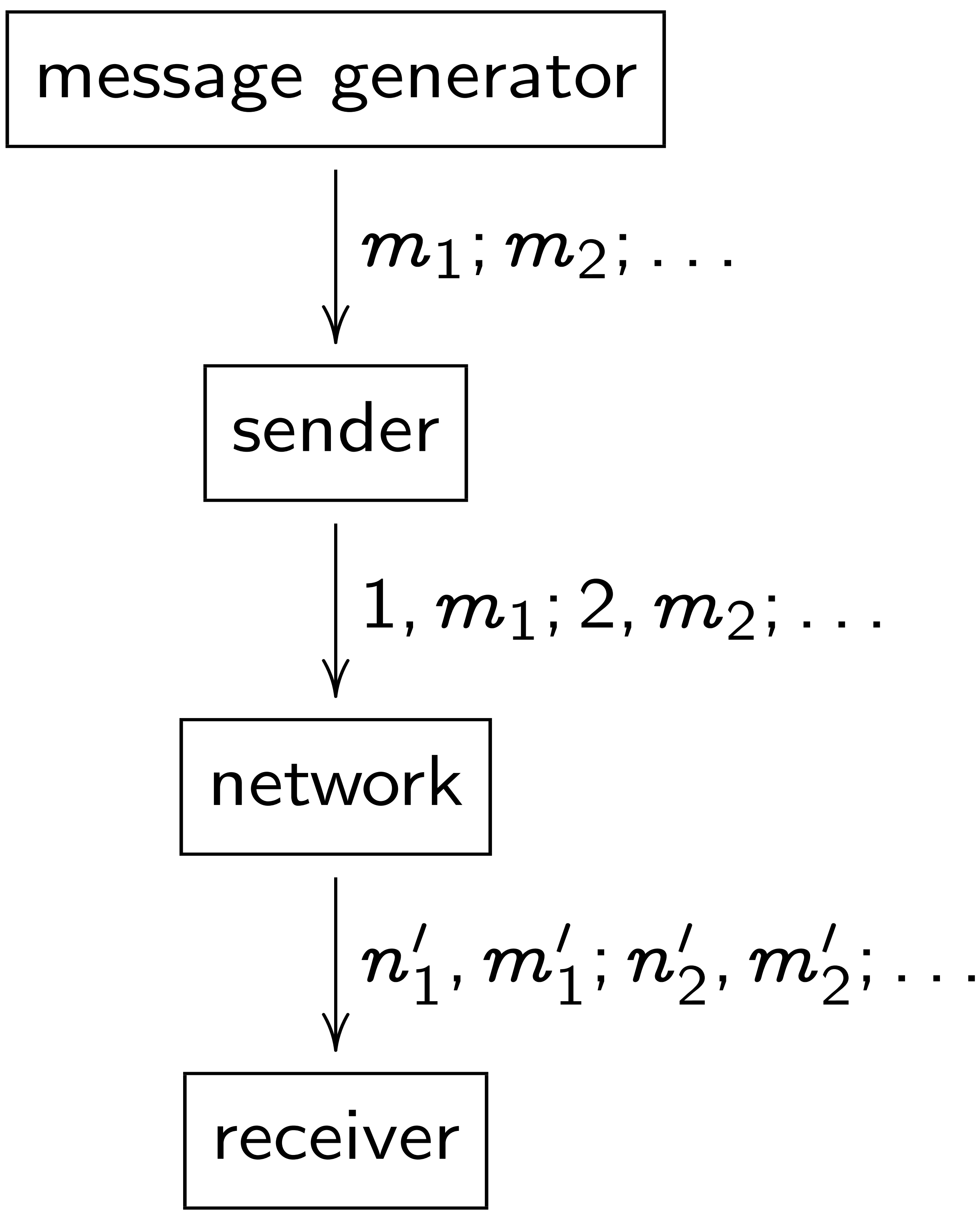
Many messages, unprotected:

$$\begin{tikzcd}
\boxed{\text{message generator}} \arrow[d, "m_1; m_2; \ldots"] \\
\boxed{\text{sender}} \arrow[d, "1, m_1; 2, m_2; \ldots"] \\
\boxed{\text{network}} \arrow[d, "n'_1, m'_1; n'_2, m'_2; \ldots"] \\
\boxed{\text{receiver}}
\end{tikzcd}$$

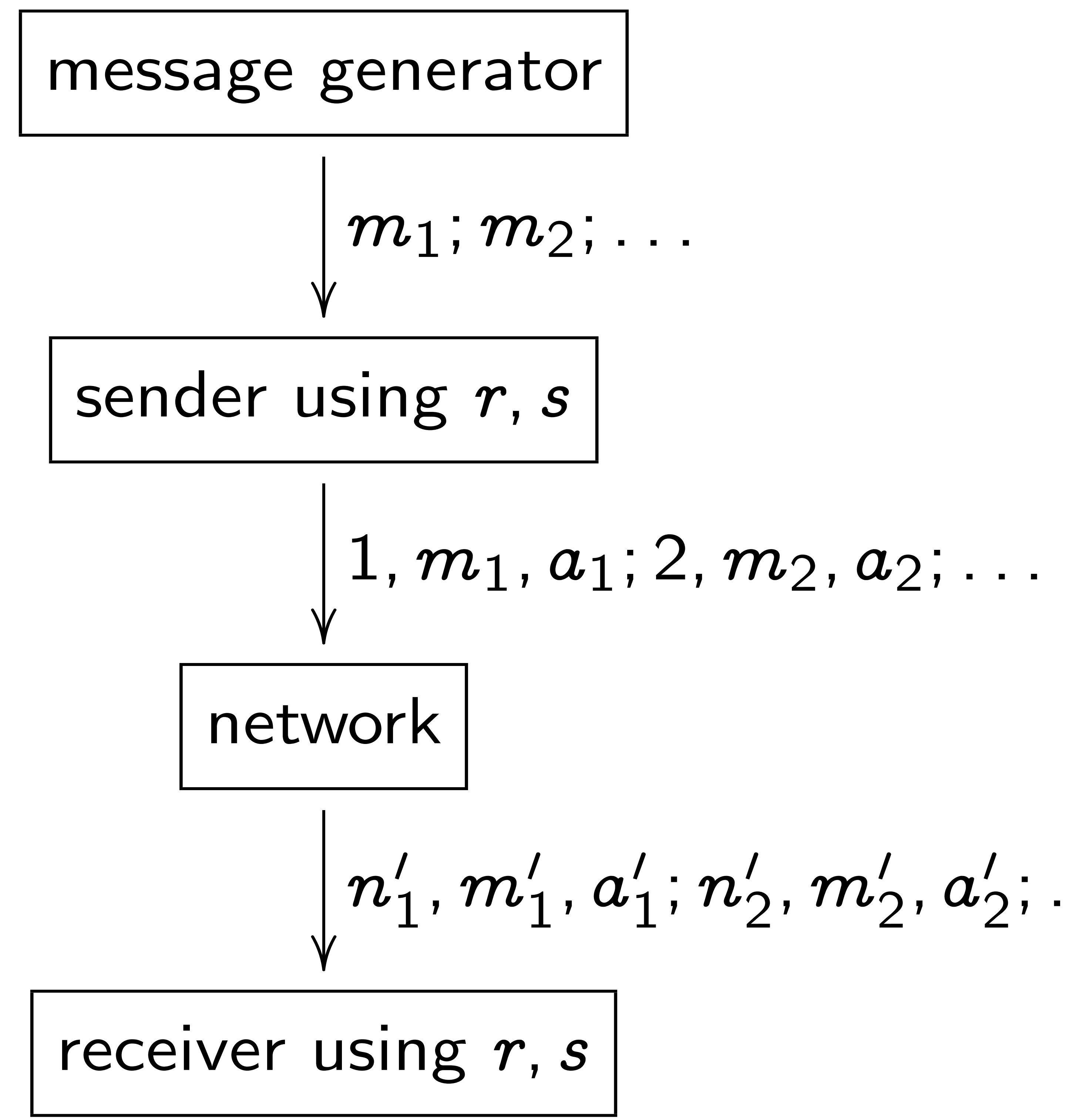
Many messages,

$$\begin{tikzcd}
\boxed{\text{message genera}} \arrow[d, "m_1; n"] \\
\boxed{\text{sender using } r} \arrow[d, "1, m_1"] \\
\boxed{\text{network}} \arrow[d, "n'_1, m"] \\
\boxed{\text{receiver using } r}
\end{tikzcd}$$

Many messages, unprotected:



Many messages, protected:



unprotected:

tor

$n_2; \ldots$

$; 2, m_2; \ldots$

$n_1'; n_2', m_2'; \ldots$

Many messages, protected:

message generator

$\downarrow$ $m_1; m_2; \ldots$

sender using r, s

$\downarrow$ $1, m_1, a_1; 2, m_2, a_2; \ldots$

network

$\downarrow$ $n_1', m_1', a_1'; n_2', m_2', a_2'; \ldots$

receiver using r, s

Secret here is un

$(r, s) \in k \times k^{\{1,2}$

i.e., $r \in k; s(1) \in$

e.g. 128000 secr

to handle 999 me

if $\#k = 2^{128}$.

Sender transmits

as $n, m, \underline{m}(r) +$

Receiver discards

if $a' \neq \underline{m'}(r) + s$

Forged n', m', a'

chance of being

Many messages, protected:

$$\begin{tikzcd}
\boxed{\text{message generator}} \arrow[d, "m_1; m_2; \ldots"] \\
\boxed{\text{sender using } r, s} \arrow[d, "1, m_1, a_1; 2, m_2, a_2; \ldots"] \\
\boxed{\text{network}} \arrow[d, "n'_1, m'_1, a'_1; n'_2, m'_2, a'_2; \ldots"] \\
\boxed{\text{receiver using } r, s}
\end{tikzcd}$$

Secret here is uniform random $(r, s) \in k \times k^{\{1,2,\ldots\}}$;
i.e., $r \in k$; $s(1) \in k$; $s(2) \in k$; ....

e.g. 128000 secret bits to handle 999 messages if $\#k = 2^{128}$.

Sender transmits nth message m as $n, m, \underline{m}(r) + s(n)$.

Receiver discards n', m', a' if $a' \neq \underline{m'}(r) + s(n')$.

Forged n', m', a' has negligible chance of being accepted.

protected:

$\boxed{\text{tor}}$

$m_2; \ldots$

$\boxed{, s}$

$, a_1; 2, m_2, a_2; \ldots$

$m'_1, a'_1; n'_2, m'_2, a'_2; \ldots$

$\boxed{r, s}$

Secret here is uniform random
$(r, s) \in k \times k^{\{1,2,\ldots\}}$;
i.e., $r \in k$; $s(1) \in k$; $s(2) \in k$; ....

e.g. 128000 secret bits
to handle 999 messages
if $\#k = 2^{128}$.

Sender transmits nth message m
as $n, m, \underline{m}(r) + s(n)$.

Receiver discards n', m', a'
if $a' \neq \underline{m'}(r) + s(n')$.

Forged n', m', a' has negligible
chance of being accepted.

How did sender,
create and share
Must have had p
providing secrecy

Why not use tha
for new messages

Answer 1: Exten
through time. Pr
can disappear aft
New channel sen

Answer 2: Expar
Messages can be
than r, s.

Secret here is uniform random $(r, s) \in k \times k^{\{1,2,\ldots\}}$;
i.e., $r \in k; s(1) \in k; s(2) \in k; \ldots$.

e.g. 128000 secret bits to handle 999 messages if $\#k = 2^{128}$.

Sender transmits nth message m as $n, m, \underline{m}(r) + s(n)$.

Receiver discards n', m', a' if $a' \neq \underline{m'}(r) + s(n')$.

Forged n', m', a' has negligible chance of being accepted.

How did sender, receiver create and share r, s? Must have had previous channel providing secrecy, authenticity.

Why not use that channel for new messages?

Answer 1: Extend security through time. Previous channel can disappear after sending r, s. New channel sends new messages.

Answer 2: Expand bandwidth. Messages can be much longer than r, s.

iform random
$2,\ldots\}$;
$\in k; s(2) \in k; \ldots$.

et bits
essages

nth message m
$s(n)$.

n', m', a'
$s(n')$.

has negligible
accepted.

How did sender, receiver
create and share r, s?
Must have had previous channel
providing secrecy, authenticity.

Why not use that channel
for new messages?

Answer 1: Extend security
through time. Previous channel
can disappear after sending r, s.
New channel sends new messages.

Answer 2: Expand bandwidth.
Messages can be much longer
than r, s.

For b-bit security
c messages, total

transmit r,
old channel
secrecy and a
for $b +$

$\Downarrow$

transmit $m_1; m$
new channe
authen
for d

How did sender, receiver create and share r, s? Must have had previous channel providing secrecy, authenticity.

Why not use that channel for new messages?

Answer 1: Extend security through time. Previous channel can disappear after sending r, s. New channel sends new messages.

Answer 2: Expand bandwidth. Messages can be much longer than r, s.

For b-bit security, $\lceil \lg \#k \rceil = b$, c messages, total length d:

transmit r, s through
old channel providing
secrecy and authenticity
for $b + bc$ bits

$$\Downarrow$$

transmit $m_1; m_2; \ldots$ through
new channel providing
authenticity
for d bits

receiver
r, s?
revious channel
, authenticity.

t channel
s?

d security
revious channel
ter sending r, s.
ds new messages.

nd bandwidth.
much longer

For b-bit security, $\lceil \lg \# k \rceil = b$,
c messages, total length d:

transmit r, s through old channel providing secrecy and authenticity for $b + bc$ bits

$$\Downarrow$$

transmit $m_1; m_2; \ldots$ through new channel providing authenticity for d bits

Authenticated-en
using $n, ((m, \underline{m}($

transmit $r,$ old channel secrecy and a for $b + bc$

$$\Downarrow$$

transmit $m_1; m$ new channe secrecy and a for d

For b-bit security, $\lceil \lg \#k \rceil = b$, c messages, total length d:

transmit r, s through old channel providing secrecy and authenticity for $b + bc$ bits

$$\Downarrow$$

transmit $m_1; m_2; \ldots$ through new channel providing authenticity for d bits

Authenticated-encryption variant using $n, ((m, \underline{m}(r)) + s(n))$:

transmit r, s through old channel providing secrecy and authenticity for $b + bc + d$ bits

$$\Downarrow$$

transmit $m_1; m_2; \ldots$ through new channel providing secrecy and authenticity for d bits

$, \lceil \lg \#k \rceil = b$,
l length d:

s through
providing
authenticity
bc bits

$m_2; \ldots$ through
l providing
nticity
bits

Authenticated-encryption variant
using $n, ((m, \underline{m}(r)) + s(n))$:

transmit r, s through
old channel providing
secrecy and authenticity
for $b + bc + d$ bits

$$\Downarrow$$

transmit $m_1; m_2; \ldots$ through
new channel providing
secrecy and authenticity
for d bits

Can multiply in k
using $b^{1+o(1)}$ bit
more precisely, $b($

Can evaluate $\underline{m}($
using $b(\lg b)^{1+o(1}$
for each b-bit blo

Overall $(bc + d)($
bit operations.

Normally d domi
so $(\lg b)^{1+o(1)}$ bi
for each message

Authenticated-encryption variant
using $n, ((m, \underline{m}(r)) + s(n))$:

transmit r, s through
old channel providing
secrecy and authenticity
for $b + bc + d$ bits

$$\Downarrow$$

transmit $m_1; m_2; \ldots$ through
new channel providing
secrecy and authenticity
for d bits

Can multiply in k
using $b^{1+o(1)}$ bit operations;
more precisely, $b(\lg b)^{1+o(1)}$.

Can evaluate $\underline{m}(r)$
using $b(\lg b)^{1+o(1)}$ bit operations
for each b-bit block of m.

Overall $(bc + d)(\lg b)^{1+o(1)}$
bit operations.

Normally d dominates bc,
so $(\lg b)^{1+o(1)}$ bit operations
for each message bit.

ncryption variant
$r)) + s(n))$:

s through
providing
authenticity
$+ d$ bits

$_2; \ldots$ through
l providing
authenticity
bits

Can multiply in k using $b^{1+o(1)}$ bit operations; more precisely, $b(\lg b)^{1+o(1)}$.

Can evaluate $\underline{m}(r)$ using $b(\lg b)^{1+o(1)}$ bit operations for each b-bit block of m.

Overall $(bc + d)(\lg b)^{1+o(1)}$ bit operations.

Normally d dominates bc, so $(\lg b)^{1+o(1)}$ bit operations for each message bit.

Can analyze cost
Can modify $\underline{m}(r$
to improve const
Can account for
Can focus on use

Speed records: P
See `cr.yp.to/m`
`papers.html#po`
128-bit coefficien
$k = \mathbf{Z}/(2^{130} - 5$
≈ 0.5 CPU cycle

Survey of alterna
8–10 of `papers.`

Can multiply in k
using $b^{1+o(1)}$ bit operations;
more precisely, $b(\lg b)^{1+o(1)}$.

Can evaluate $\underline{m}(r)$
using $b(\lg b)^{1+o(1)}$ bit operations
for each b-bit block of m.

Overall $(bc + d)(\lg b)^{1+o(1)}$
bit operations.

Normally d dominates bc,
so $(\lg b)^{1+o(1)}$ bit operations
for each message bit.

Can analyze cost more precisely.
Can modify $\underline{m}(r)$
to improve constants in cost.
Can account for real CPUs.
Can focus on useful b.

Speed records: Poly1305.
See `cr.yp.to/mac.html` and
`papers.html#poly1305`.
128-bit coefficients of $\underline{m}$;
$k = \mathbf{Z}/(2^{130} - 5)$; restricted r;
≈ 0.5 CPU cycles per bit.

Survey of alternatives: Sections
8–10 of `papers.html#hash127`.

k

operations;

$(\lg b)^{1+o(1)}$.

$r)$

bit operations

ock of m.

$\lg b)^{1+o(1)}$

nates bc,

t operations

bit.

Can analyze cost more precisely.
Can modify $\underline{m}(r)$
to improve constants in cost.
Can account for real CPUs.
Can focus on useful b.

Speed records: Poly1305.
See `cr.yp.to/mac.html` and
`papers.html#poly1305`.
128-bit coefficients of $\underline{m}$;
$k = \mathbf{Z}/(2^{130} - 5)$; restricted r;
$\approx$ 0.5 CPU cycles per bit.

Survey of alternatives: Sections
8–10 of `papers.html#hash127`.

We'll see better

that dramatically

requirements on

1. Reduce bandw

far below $b + bc$

2. Eliminate secr

Disadvantage: n

prove security of

Can analyze cost more precisely.
Can modify $\underline{m}(r)$
to improve constants in cost.
Can account for real CPUs.
Can focus on useful b.

Speed records: Poly1305.
See `cr.yp.to/mac.html` and
`papers.html#poly1305`.
128-bit coefficients of $\underline{m}$;
$k = \mathbf{Z}/(2^{130} - 5)$; restricted r;
≈ 0.5 CPU cycles per bit.

Survey of alternatives: Sections
8–10 of `papers.html#hash127`.

We'll see better protocols
that dramatically reduce
requirements on old channel:

1. Reduce bandwidth
far below $b + bc + d$ bits.

2. Eliminate secrecy.

Disadvantage: no known way to
prove security of better protocols.

more precisely.

)

ants in cost.

real CPUs.

eful b.

Poly1305.

ac.html and

oly1305.

nts of $\underline{m}$;

); restricted r;

s per bit.

tives: Sections

html#hash127.

We'll see better protocols that dramatically reduce requirements on old channel:

1. Reduce bandwidth far below $b + bc + d$ bits.

2. Eliminate secrecy.

Disadvantage: no known way to *prove* security of better protocols.

How to reduce ba

Expand a short s

into a long share

e.g. Expand t, u_0

$4^t \bmod q$,

$4^{tu_0} \bmod q$,

$4^{tu_1} \bmod q$,

$4^{tu_0 u_1} \bmod q$,

$4^{tu_2} \bmod q$,

$4^{tu_0 u_2} \bmod q$,

$4^{tu_1 u_2} \bmod q$,

$4^{tu_0 u_1 u_2} \bmod q$,

etc., where $q = 2$

We'll see better protocols that dramatically reduce requirements on old channel:

1. Reduce bandwidth far below $b + bc + d$ bits.

2. Eliminate secrecy.

Disadvantage: no known way to *prove* security of better protocols.

How to reduce bandwidth?

Expand a short shared secret into a long shared secret.

e.g. Expand $t, u_0, u_1, \ldots$ into
$4^t \bmod q$,
$4^{tu_0} \bmod q$,
$4^{tu_1} \bmod q$,
$4^{tu_0u_1} \bmod q$,
$4^{tu_2} \bmod q$,
$4^{tu_0u_2} \bmod q$,
$4^{tu_1u_2} \bmod q$,
$4^{tu_0u_1u_2} \bmod q$,
etc., where $q = 2^{2000} - 1553657$.

protocols
reduce
old channel:

width
$+ d$ bits.

recy.

o known way to
better protocols.

How to reduce bandwidth?

Expand a short shared secret into a long shared secret.

e.g. Expand $t, u_0, u_1, \ldots$ into
$4^t \bmod q$,
$4^{tu_0} \bmod q$,
$4^{tu_1} \bmod q$,
$4^{tu_0u_1} \bmod q$,
$4^{tu_2} \bmod q$,
$4^{tu_0u_2} \bmod q$,
$4^{tu_1u_2} \bmod q$,
$4^{tu_0u_1u_2} \bmod q$,
etc., where $q = 2^{2000} - 1553657$.

Conjecture: Hard
this expanded se
from a uniform r
of squares modul

Could try to do i
discrete logs mod
extract t from 4^t
tu_0 from 4^{tu_0} m
and see $(t)(tu_0u$
But discrete logs

Thus (e.g.) botto
seem hard to dist
from a uniform r

How to reduce bandwidth?

Expand a short shared secret into a long shared secret.

e.g. Expand $t, u_0, u_1, \ldots$ into
$4^t \bmod q$,
$4^{tu_0} \bmod q$,
$4^{tu_1} \bmod q$,
$4^{tu_0u_1} \bmod q$,
$4^{tu_2} \bmod q$,
$4^{tu_0u_2} \bmod q$,
$4^{tu_1u_2} \bmod q$,
$4^{tu_0u_1u_2} \bmod q$,
etc., where $q = 2^{2000} - 1553657$.

Conjecture: Hard to distinguish this expanded secret from a uniform random sequence of squares modulo q.

Could try to do it by computing discrete logs modulo q:
extract t from $4^t \bmod q$,
tu_0 from $4^{tu_0} \bmod q$, etc.,
and see $(t)(tu_0u_1) = (tu_0)(tu_1)$.
But discrete logs seem hard!

Thus (e.g.) bottom halves seem hard to distinguish from a uniform random string.

andwidth?

hared secret

d secret.

$_0, u_1, \ldots$ into

$2^{2000} - 1553657.$

Conjecture: Hard to distinguish this expanded secret from a uniform random sequence of squares modulo q.

Could try to do it by computing discrete logs modulo q: extract t from $4^t \bmod q$, tu_0 from $4^{tu_0} \bmod q$, etc., and see $(t)(tu_0u_1) = (tu_0)(tu_1)$. But discrete logs seem hard!

Thus (e.g.) bottom halves seem hard to distinguish from a uniform random string.

For b bits of secu

Fix q with $q, (q -$

and with $\lg q \in b$

more precisely, w

$6.8 \ldots (\lg q)(\lg \mathrm{lo}$

Transmit short sh

independent unif

$t, u_0, u_1, \ldots \in \{1$

These sizes just

fastest discrete-lo

that we know.

Conjecture: Hard to distinguish this expanded secret from a uniform random sequence of squares modulo q.

Could try to do it by computing discrete logs modulo q: extract t from $4^t \bmod q$, tu_0 from $4^{tu_0} \bmod q$, etc., and see $(t)(tu_0u_1) = (tu_0)(tu_1)$. But discrete logs seem hard!

Thus (e.g.) bottom halves seem hard to distinguish from a uniform random string.

For b bits of security, $b \to \infty$:

Fix q with $q, (q-1)/2$ prime and with $\lg q \in b^{3+o(1)}$; more precisely, with $6.8\ldots(\lg q)(\lg \log q)^2 \approx b^3$.

Transmit short shared secret: independent uniform random $t, u_0, u_1, \ldots \in \{1, 2, \ldots, 2^{2b}\}$.

These sizes just barely resist fastest discrete-log methods that we know.

d to distinguish
cret
andom sequence
lo q.

t by computing
dulo q:
mod q,
od q, etc.,
$_1) = (tu_0)(tu_1)$.
seem hard!

om halves
tinguish
andom string.

For b bits of security, $b \to \infty$:

Fix q with $q, (q-1)/2$ prime and with $\lg q \in b^{3+o(1)}$; more precisely, with $6.8\ldots(\lg q)(\lg\log q)^2 \approx b^3$.

Transmit short shared secret: independent uniform random $t, u_0, u_1, \ldots \in \{1, 2, \ldots, 2^{2b}\}$.

These sizes just barely resist fastest discrete-log methods that we know.

Expand $t, u_0, u_1,$
$4^t \bmod q, \ldots, 4^{t}$

e.g. Expand t, u
into 2^{64} integers

Extract bottom
of each integer.

Compute results
Only $O(b)$ mults
for each integer,
$b(\lg b)^{1+o(1)}$ bit

Random access i
$\geq b^{4+o(1)}$ bit ops

For b bits of security, $b \to \infty$:

Fix q with $q, (q-1)/2$ prime and with $\lg q \in b^{3+o(1)}$; more precisely, with $6.8\ldots(\lg q)(\lg\log q)^2 \approx b^3$.

Transmit short shared secret: independent uniform random $t, u_0, u_1, \ldots \in \{1, 2, \ldots, 2^{2b}\}$.

These sizes just barely resist fastest discrete-log methods that we know.

Expand $t, u_0, u_1, \ldots$ into $4^t \bmod q, \ldots, 4^{tu_0u_2} \bmod q, \ldots$.

e.g. Expand $t, u_0, u_1, \ldots, u_{63}$ into 2^{64} integers modulo q.

Extract bottom $\lceil (1/2)\lg q \rceil$ bits of each integer.

Compute results sequentially. Only $O(b)$ mults mod q for each integer, so $b(\lg b)^{1+o(1)}$ bit ops per bit.

Random access is slow: $\geq b^{4+o(1)}$ bit ops.

urity, $b \to \infty$:

$-1)/2$ prime
$^{3+o(1)}$;

vith
$\mathrm{g}\, q)^2 \approx b^3$.

hared secret:
orm random
$1, 2, \ldots, 2^{2b}\}$.

barely resist
og methods

Expand $t, u_0, u_1, \ldots$ into
$4^t \bmod q, \ldots, 4^{t u_0 u_2} \bmod q, \ldots$.

e.g. Expand $t, u_0, u_1, \ldots, u_{63}$
into 2^{64} integers modulo q.

Extract bottom $\lceil (1/2) \lg q \rceil$ bits
of each integer.

Compute results sequentially.
Only $O(b)$ mults mod q
for each integer, so
$b(\lg b)^{1+o(1)}$ bit ops per bit.

Random access is slow:
$\geq b^{4+o(1)}$ bit ops.

Do better by rep
$(\mathbf{Z}/q)^*$ with $E(\mathbf{Z}$
for a safe elliptic

Discrete logs in E
seem relatively d
so can take q sm
specifically, $\lg q \approx$

Much faster ranc
$b^{2+o(1)}$ bit ops.

Sequential access
$b(\lg b)^{1+o(1)}$ bit o

Expand $t, u_0, u_1, \ldots$ into $4^t \bmod q, \ldots, 4^{tu_0u_2} \bmod q, \ldots$.

e.g. Expand $t, u_0, u_1, \ldots, u_{63}$ into 2^{64} integers modulo q.

Extract bottom $\lceil (1/2) \lg q \rceil$ bits of each integer.

Compute results sequentially. Only $O(b)$ mults mod q for each integer, so $b(\lg b)^{1+o(1)}$ bit ops per bit.

Random access is slow: $\geq b^{4+o(1)}$ bit ops.

Do better by replacing $(\mathbf{Z}/q)^*$ with $E(\mathbf{Z}/q)$ for a safe elliptic curve E.

Discrete logs in $E(\mathbf{Z}/q)$ seem relatively difficult, so can take q smaller: specifically, $\lg q \approx 2b$.

Much faster random access: $b^{2+o(1)}$ bit ops.

Sequential access again takes $b(\lg b)^{1+o(1)}$ bit ops per bit.

. . . into
$u_0 u_2 \bmod q, \ldots.$

$_0, u_1, \ldots, u_{63}$
modulo q.

$\lceil(1/2)\lg q\rceil$ bits

sequentially.
mod q
so
ops per bit.

s slow:
s.

Do better by replacing $(\mathbf{Z}/q)^*$ with $E(\mathbf{Z}/q)$ for a safe elliptic curve E.

Discrete logs in $E(\mathbf{Z}/q)$ seem relatively difficult, so can take q smaller: specifically, $\lg q \approx 2b$.

Much faster random access: $b^{2+o(1)}$ bit ops.

Sequential access again takes $b(\lg b)^{1+o(1)}$ bit ops per bit.

Can do much bet
Maybe non-const
certainly constan
just one b-bit sec
several $2b$-bit sec
focus on useful b

Many choices of
functions ("strea
Fastest expansion
don't have discre

Speed records: se
`www.ecrypt.eu`
Often $<$ 1 CPU c
random access $<$

Do better by replacing $(\mathbf{Z}/q)^*$ with $E(\mathbf{Z}/q)$ for a safe elliptic curve E.

Discrete logs in $E(\mathbf{Z}/q)$ seem relatively difficult, so can take q smaller: specifically, $\lg q \approx 2b$.

Much faster random access: $b^{2+o(1)}$ bit ops.

Sequential access again takes $b(\lg b)^{1+o(1)}$ bit ops per bit.

Can do much better.
Maybe non-constant speedups; certainly constant speedups; just one b-bit secret instead of several $2b$-bit secrets $t, u_0, u_1, \ldots$; focus on useful b; etc.

Many choices of expansion functions ("stream ciphers").
Fastest expansion functions *don't* have discrete-log structure.

Speed records: see eSTREAM, `www.ecrypt.eu.org/stream`.
Often < 1 CPU cycle per bit; random access < 1000 cycles.

lacing
$/q)$
curve E.

$E(\mathbf{Z}/q)$
ifficult,
aller:
$\approx 2b$.

dom access:

s again takes
ops per bit.

Can do much better.
Maybe non-constant speedups;
certainly constant speedups;
just one b-bit secret instead of
several $2b$-bit secrets $t, u_0, u_1, \ldots$;
focus on useful b; etc.

Many choices of expansion
functions ("stream ciphers").
Fastest expansion functions
don't have discrete-log structure.

Speed records: see eSTREAM,
`www.ecrypt.eu.org/stream`.
Often < 1 CPU cycle per bit;
random access < 1000 cycles.

Channels at this

transmit sec
old channel
secrecy and
for b

$\Downarrow$

transmit $m_1; m$
new channe
secrecy and
for d

Can do much better.

Maybe non-constant speedups; certainly constant speedups; just one b-bit secret instead of several $2b$-bit secrets $t, u_0, u_1, \ldots$; focus on useful b; etc.

Many choices of expansion functions ("stream ciphers"). Fastest expansion functions *don't* have discrete-log structure.

Speed records: see eSTREAM, `www.ecrypt.eu.org/stream`. Often < 1 CPU cycle per bit; random access < 1000 cycles.

Channels at this point:

transmit secret through old channel providing secrecy and authenticity for b bits

$$\Downarrow$$

transmit $m_1; m_2; \ldots$ through new channel providing secrecy and authenticity for d bits tter.

tant speedups;

t speedups;

cret instead of

crets $t, u_0, u_1, \ldots$;

; etc.

expansion

m ciphers").

n functions

ete-log structure.

ee eSTREAM,

`.org/stream`.

cycle per bit;

1000 cycles.

Channels at this point:

transmit secret through old channel providing secrecy and authenticity for b bits

$\Downarrow$

transmit $m_1; m_2; \ldots$ through new channel providing secrecy and authenticity for d bits

How to use old c

providing only au

not secrecy?

Sender generates

sends **public key**

through old chan

Receiver generate

sends public key

back through old

Sender and recei

compute $4^{\sigma\tau}$ mo

extract b bits,

expand into long

Channels at this point:

transmit secret through old channel providing secrecy and authenticity for b bits

$$\Downarrow$$

transmit $m_1; m_2; \ldots$ through new channel providing secrecy and authenticity for d bits

How to use old channel providing only authenticity, not secrecy?

Sender generates secret σ, sends **public key** $4^\sigma \bmod q$ through old channel.

Receiver generates secret τ, sends public key $4^\tau \bmod q$ back through old channel.

Sender and receiver now compute $4^{\sigma\tau} \bmod q$,
extract b bits,
expand into long shared secret.

point:

ret through
providing
authenticity
bits

,

$a_2; \ldots$ through
l providing
authenticity
bits

How to use old channel providing only authenticity, not secrecy?

Sender generates secret σ, sends **public key** $4^\sigma \bmod q$ through old channel.

Receiver generates secret τ, sends public key $4^\tau \bmod q$ back through old channel.

Sender and receiver now compute $4^{\sigma\tau} \bmod q$, extract b bits, expand into long shared secret.

Conjecture: Give
$4^\tau \bmod q$, hard t
$4^{\sigma\tau} \bmod q$ from
uniform random

As before, can di
by computing dis
but that seems h

As before, reduce
by switching to e

Many choices of
No! Need discret

How to use old channel providing only authenticity, not secrecy?

Sender generates secret σ, sends **public key** $4^\sigma \bmod q$ through old channel.

Receiver generates secret τ, sends public key $4^\tau \bmod q$ back through old channel.

Sender and receiver now compute $4^{\sigma\tau} \bmod q$, extract b bits, expand into long shared secret.

Conjecture: Given $4^\sigma \bmod q$, $4^\tau \bmod q$, hard to distinguish $4^{\sigma\tau} \bmod q$ from uniform random square mod q.

As before, can distinguish by computing discrete logs, but that seems hard.

As before, reduce costs by switching to elliptic curves.

Many choices of functions? No! Need discrete-log structure.

channel
uthenticity,

s secret σ,
y 4^σ mod q
nel.

es secret τ,
4^τ mod q
l channel.

ver now
d q,

shared secret.

Conjecture: Given 4^σ mod q, 4^τ mod q, hard to distinguish $4^{\sigma\tau}$ mod q from uniform random square mod q.

As before, can distinguish by computing discrete logs, but that seems hard.

As before, reduce costs by switching to elliptic curves.

Many choices of functions? No! Need discrete-log structure.

Again improve co
focus on useful b

Speed records: C
See `cr.yp.to/e`
`papers.html#cu`
Curve $y^2 = x^3 +$
mod $2^{255} - 19$; e
< 1000000 CPU

New records fron
of genus-2 hyper
Come to ECC 20
`www.fields.uto`
`/programs/scie`
`/06-07/crypto/`

Conjecture: Given $4^\sigma \bmod q$, $4^\tau \bmod q$, hard to distinguish $4^{\sigma\tau} \bmod q$ from uniform random square mod q.

As before, can distinguish by computing discrete logs, but that seems hard.

As before, reduce costs by switching to elliptic curves.

Many choices of functions? No! Need discrete-log structure.

Again improve cost constants, focus on useful b, etc.

Speed records: Curve25519. See `cr.yp.to/ecdh.html` and `papers.html#curve25519`. Curve $y^2 = x^3 + 486662x^2 + x$ mod $2^{255} - 19$; eliminate y; $<$ 1000000 CPU cycles.

New records from Jacobians of genus-2 hyperelliptic curves? Come to ECC 2006 in Toronto. `www.fields.utoronto.ca` `/programs/scientific` `/06-07/crypto/`

en 4^σ mod q,
o distinguish

square mod q.

stinguish
screte logs,
ard.

e costs
elliptic curves.

functions?
te-log structure.

Again improve cost constants, focus on useful b, etc.

Speed records: Curve25519. See `cr.yp.to/ecdh.html` and `papers.html#curve25519`. Curve $y^2 = x^3 + 486662x^2 + x$ mod $2^{255} - 19$; eliminate y; < 1000000 CPU cycles.

New records from Jacobians of genus-2 hyperelliptic curves? Come to ECC 2006 in Toronto. `www.fields.utoronto.ca /programs/scientific /06-07/crypto/`

Many other publ

e.g. "Public-key
reduces costs of
sending a public
to many recipien

Sender signs mes
independently of
without any shar

Each receiver ver
very fast comput

Again improve cost constants, focus on useful b, etc.

Speed records: Curve25519. See `cr.yp.to/ecdh.html` and `papers.html#curve25519`. Curve $y^2 = x^3 + 486662x^2 + x$ mod $2^{255} - 19$; eliminate y; < 1000000 CPU cycles.

New records from Jacobians of genus-2 hyperelliptic curves? Come to ECC 2006 in Toronto. `www.fields.utoronto.ca/programs/scientific/06-07/crypto/`

Many other public-key structures.

e.g. "Public-key signing" reduces costs of securely sending a public message to many recipients.

Sender signs message independently of receiver, without any shared secrets.

Each receiver verifies signature; very fast computation.

ost constants,
, etc.

Curve25519.
cdh.html and
urve25519.
$486662x^2 + x$
eliminate y;
cycles.

n Jacobians
elliptic curves?
006 in Toronto.
oronto.ca
entific
/

Many other public-key structures.

e.g. "Public-key signing"
reduces costs of securely
sending a public message
to many recipients.

Sender signs message
independently of receiver,
without any shared secrets.

Each receiver verifies signature;
very fast computation.

e.g. "Public-key
generates new sh
for each message

Always increases
as far as I know.
The literature on
"public-key encry
is a historical acc

Best to reuse one
for many messag
Minimize public-

Many other public-key structures.

e.g. “Public-key signing” reduces costs of securely sending a public message to many recipients.

Sender signs message independently of receiver, without any shared secrets.

Each receiver verifies signature; very fast computation.

e.g. “Public-key encryption” generates new shared secret for each message.

Always increases costs, as far as I know.
The literature on “public-key encryption” is a historical accident.

Best to reuse one secret for many messages.
Minimize public-key operations.

ic-key structures.

signing"
securely
message
ts.

ssage
receiver,
ed secrets.

rifies signature;
ation.

e.g. "Public-key encryption"
generates new shared secret
for each message.

Always increases costs,
as far as I know.
The literature on
"public-key encryption"
is a historical accident.

Best to reuse one secret
for many messages.
Minimize public-key operations.

If large quantum
are built then the
compute discrete
Huge effects on c

Some public-key
survive quantum
See PQCrypto 20
`postquantum.cr`

Exactly how fast
DSA, ECDH, pos
cryptosystems, et
`www.ecrypt.eu`
benchmarking pu

e.g. “Public-key encryption” generates new shared secret for each message.

Always increases costs, as far as I know. The literature on “public-key encryption” is a historical accident.

Best to reuse one secret for many messages. Minimize public-key operations.

If large quantum computers are built then they will compute discrete logs quickly. Huge effects on cryptography!

Some public-key systems seem to survive quantum computers. See PQCrypto 2006 abstracts: `postquantum.cr.yp.to`.

Exactly how fast are RSA, DSA, ECDH, post-quantum cryptosystems, etc.? `www.ecrypt.eu.org/ebats` is benchmarking public-key systems.