Making sure crypto stays insecure

Daniel J. Bernstein
University of Illinois at Chicago & Technische Universiteit Eindhoven

Terrorist in Hong Kong prepares to throw deadly weapon at Chinese government workers.

Image credit: Reuters.
Making sure crypto stays insecure

Daniel J. Bernstein
University of Illinois at Chicago & Technische Universiteit Eindhoven

Terrorist in Hong Kong prepares to throw deadly weapon at Chinese government workers.

Image credit: Reuters.

Drug-dealing cartel "Starbucks" invades city in Morocco; begins selling addictive liquid.

Image credit: Wikipedia.
Making sure crypto stays insecure
Daniel J. Bernstein
University of Illinois at Chicago & Technische Universiteit Eindhoven

Terrorist in Hong Kong prepares to throw deadly weapon at Chinese government workers.
Image credit: Reuters.

Drug-dealing cartel "Starbucks" invades city in Morocco; begins selling addictive liquid.
Image credit: Wikipedia.
Terrorist in Hong Kong prepares to throw deadly weapon at Chinese government workers.

Image credit: Reuters.

Drug-dealing cartel “Starbucks” invades city in Morocco; begins selling addictive liquid.

Image credit: Wikipedia.
Terrorist in Hong Kong prepares to throw deadly weapon at Chinese government workers.

Image credit: Reuters.

Drug-dealing cartel “Starbucks” invades city in Morocco; begins selling addictive liquid.

Image credit: Wikipedia.
Terrorist in Hong Kong prepares to throw deadly weapon at Chinese government workers.

Drug-dealing cartel "Starbucks" invades city in Morocco; begins selling addictive liquid.

Pedophile convinces helpless child to remove most of her clothing; sexually abuses child in public.

Image credit: Reuters.

Image credit: Wikipedia.

Image credit: Child pornographer.
Terrorist in Hong Kong prepares to throw deadly weapon at Chinese government workers.

Image credit: Reuters.

Drug-dealing cartel “Starbucks” invades city in Morocco; begins selling addictive liquid.

Image credit: Wikipedia.

Pedophile convinces helpless child to remove most of her clothing; sexually abuses child in public.

Image credit: Child pornographer.
Terrorist in Hong Kong prepares to throw deadly weapon at Chinese government workers.

Image credit: Reuters.

Drug-dealing cartel “Starbucks” invades city in Morocco; begins selling addictive liquid.

Image credit: Wikipedia.

Pedophile convinces helpless child to remove most of her clothing; sexually abuses child in public.

Image credit: Child pornography.
Drug-dealing cartel “Starbucks” invades city in Morocco; begins selling addictive liquid.

Image credit: Wikipedia.

Pedophile convinces helpless child to remove most of her clothing; sexually abuses child in public.

Image credit: Child pornographer.
Drug-dealing cartel “Starbucks” invades city in Morocco; begins selling addictive liquid.

Image credit: Wikipedia.

Pedophile convinces helpless child to remove most of her clothing; sexually abuses child in public.

Image credit: Child pornographer.

Criminal organization calling itself “The Guardian” sells classified government secrets.

Image credit: The Guardian.
Drug-dealing cartel "Starbucks" invades city in Morocco; begins selling addictive liquid.

Image credit: Wikipedia.

Pedophile convinces helpless child to remove most of her clothing; sexually abuses child in public.

Image credit: Child pornographer.

Criminal organization calling itself "The Guardian" sells classified government secrets.

Image credit: The Guardian.
Pedophile convinces helpless child to remove most of her clothing; sexually abuses child in public.

Image credit: Child pornographer.
Pedophile convinces helpless child to remove most of her clothing; sexually abuses child in public.

Image credit: Child pornographer.

Criminal organization calling itself “The Guardian” sells classified government secrets.

Image credit: The Guardian.
We have to watch and listen to everything that people are doing, so that we can catch terrorists, drug dealers, organized criminals, pedophiles, etc.
We have to watch and listen to everything that people are doing so that we can catch terrorists, drug dealers, organized criminals, pedophiles, murderers, etc.
Pedophile convinces helpless child to remove most of her clothing; sexually abuses child in public.

Image credit: Child pornographer.

Criminal organization calling itself “The Guardian” sells classified government secrets.

Image credit: The Guardian.
Criminal organization calling itself “The Guardian” sells classified government secrets.

Image credit: The Guardian.

We have to watch and listen to everything that people are doing so that we can catch terrorists, drug dealers, organized criminals, pedophiles, murderers, etc.
Criminal organization calling itself “The Guardian” sells classified government secrets.

Image credit: The Guardian.

We have to watch and listen to everything that people are doing so that we can catch terrorists, drug dealers, organized criminals, pedophiles, murderers, etc.

We try to systematically monitor and record all Internet traffic. But what if it’s encrypted?
Criminal organization calling itself “The Guardian” sells classified government secrets.

Image credit: The Guardian.

We have to watch and listen to everything that people are doing so that we can catch terrorists, drug dealers, organized criminals, pedophiles, murderers, etc.

We try to systematically monitor and record all Internet traffic. But what if it’s encrypted?

This talk gives some examples of how we’ve manipulated the world’s crypto ecosystem so that we can understand almost all of this traffic.
We have to watch and listen to everything that people are doing so that we can catch terrorists, drug dealers, organized criminals, pedophiles, murderers, etc.

We try to systematically monitor and record all Internet traffic. But what if it’s encrypted?

This talk gives some examples of how we’ve manipulated the world’s crypto ecosystem so that we can understand almost all of this traffic.
We have to watch and listen to everything that people are doing so that we can catch terrorists, drug dealers, organized criminals, pedophiles, murderers, etc.

We try to systematically monitor and record all Internet traffic. But what if it’s encrypted?

This talk gives some examples of how we’ve manipulated the world’s crypto ecosystem so that we can understand almost all of this traffic.
We have to watch and listen to everything that people are doing so that we can catch terrorists, drug dealers, organized criminals, pedophiles, murderers, etc.

We try to systematically monitor and record all Internet traffic. **But what if it’s encrypted?**

This talk gives some examples of how we’ve manipulated the world’s crypto ecosystem so that we can understand almost all of this traffic.
We have to watch and listen to everything that people are doing so that we can catch terrorists, drug dealers, organized criminals, pedophiles, murderers, etc.

We try to systematically monitor and record all Internet traffic. **But what if it’s encrypted?**

This talk gives some examples of how we’ve manipulated the world’s crypto ecosystem so that we can understand almost all of this traffic.
We have to watch and listen to everything that people are doing so that we can catch terrorists, drug dealers, organized criminals, pedophiles, murderers, etc.

We try to systematically monitor and record all Internet traffic. But what if it’s encrypted?

This talk gives some examples of how we’ve manipulated the world’s crypto ecosystem so that we can understand almost all of this traffic.

Other useful strategies, not covered in this talk:
Manipulate software ecosystem so that software stays insecure.
Break into computers; access hundreds of millions of disks, screens, microphones, cameras.
We have to watch and listen to everything that people are doing so that we can catch terrorists, drug dealers, organized criminals, pedophiles, murderers, etc.

We try to systematically monitor and record all Internet traffic. But what if it's encrypted?

This talk gives some examples of how we've manipulated the world's crypto ecosystem so that we can understand almost all of this traffic.

Other useful strategies, not covered in this talk:

- Manipulate software ecosystem so that software stays insecure.
- Break into computers; access hundreds of millions of disks, screens, microphones, etc.
We have to watch and listen to everything that people are doing so that we can catch terrorists, drug dealers, organized criminals, pedophiles, murderers, etc.

We try to systematically monitor and record all Internet traffic.

But what if it's encrypted?

This talk gives some examples of how we've manipulated the world's crypto ecosystem so that we can understand almost all of this traffic.

Other useful strategies, not covered in this talk:

Manipulate software ecosystem so that software stays insecure.

Break into computers; access hundreds of millions of disks, screens, microphones, cameras.
Other useful strategies, not covered in this talk:

Manipulate *software* ecosystem so that software stays insecure. Break into computers; access hundreds of millions of disks, screens, microphones, cameras.
Other useful strategies, not covered in this talk:

Manipulate *software* ecosystem so that software stays insecure.

Break into computers; access hundreds of millions of disks, screens, microphones, cameras.

Add back doors to *hardware*.

e.g. 2012 U.S. government report says that Chinese-manufactured routers provide “Chinese intelligence services access to telecommunication networks”.

* (TS/SL/REL TO USA, FVEY) Insert vulnerabilities into commercial encryption systems, IT systems, networks, and endpoint communications devices used by targets.
* (TS/SL/REL TO USA, FVEY) Collect target network data and metadata via cooperative network carriers and/or increased control over core networks.
* (TS/SL/REL TO USA, FVEY) Leverage commercial capabilities to remotely deliver or receive information to and from target endpoints.
* (TS/SL/REL TO USA, FVEY) Exploit foreign trusted computing platforms and technologies.
* (TS/SL/REL TO USA, FVEY) Influence policies, standards and specification for commercial public key technologies.
* (TS/SL/REL TO USA, FVEY) Make specific and aggressive investments to facilitate the development of a robust exploitation capability against Next-Generation Wireless (NGW) communications.
Other useful strategies, not covered in this talk:
Manipulate *software* ecosystem so that software stays insecure.
Break into computers; access hundreds of millions of disks, screens, microphones, cameras.
Add back doors to *hardware*. e.g. 2012 U.S. government report says that Chinese-manufactured routers provide “Chinese intelligence services access to telecommunication networks”.

---

1. “We” doesn’t include me.
I want secure crypto.
Other useful strategies, not covered in this talk:

Manipulate *software* ecosystem so that software stays insecure.

Break into computers; access hundreds of millions of disks, screens, microphones, cameras.

Add back doors to *hardware*.

E.g. 2012 U.S. government report says that Chinese-manufactured routers provide “Chinese intelligence services access to telecommunication networks”.

Some important clarifications

1. “We” doesn’t include me.

I want secure crypto.
Other useful strategies, not covered in this talk:

Manipulate *software* ecosystem so that software stays insecure. Break into computers; access hundreds of millions of disks, screens, microphones, cameras.

Add back doors to *hardware*. e.g. 2012 U.S. government report says that Chinese-manufactured routers provide “Chinese intelligence services access to telecommunication networks”.

Some important clarifications

1. “We” doesn’t include me. I want secure crypto.
Other useful strategies, not covered in this talk:
Manipulate software ecosystem so that software stays insecure.
Break into computers; access hundreds of millions of disks, screens, microphones, cameras.
Add back doors to hardware.
e.g. 2012 U.S. government report says that Chinese-manufactured routers provide “Chinese intelligence services access to telecommunication networks”.

Some important clarifications
1. “We” doesn’t include me.
I want secure crypto.
Other useful strategies, not covered in this talk:
Manipulate *software* ecosystem so that software stays insecure.
Break into computers; access hundreds of millions of disks, screens, microphones, cameras.
Add back doors to *hardware*.
e.g. 2012 U.S. government report says that Chinese-manufactured routers provide “Chinese intelligence services access to telecommunication networks”.

Some important clarifications
1. “We” doesn’t include me. I want secure crypto.
2. Their actions violate fundamental human rights.
Other useful strategies, not covered in this talk:

Manipulate *software* ecosystem so that software stays insecure.

Break into computers; access hundreds of millions of disks, screens, microphones, cameras.

Add back doors to *hardware*.

e.g. 2012 U.S. government report says that Chinese-manufactured routers provide “Chinese intelligence services access to telecommunication networks”.

Some important clarifications

1. “We” doesn’t include me.
   I want secure crypto.

2. Their actions violate fundamental human rights.

3. I don’t know how much of today’s crypto ecosystem was deliberately manipulated.
Other useful strategies, not covered in this talk:
Manipulate software ecosystem so that software stays insecure. Break into computers; access hundreds of millions of disks, screens, microphones, cameras.
Add back doors to hardware. e.g. 2012 U.S. government report says that Chinese-manufactured routers provide “Chinese intelligence services access to telecommunication networks”.

Some important clarifications
1. “We” doesn’t include me. I want secure crypto.
2. Their actions violate fundamental human rights.
3. I don’t know how much of today’s crypto ecosystem was deliberately manipulated.

This talk is actually a thought experiment: how could an attacker manipulate the ecosystem for insecurity?
Other useful strategies, not covered in this talk:

- Manipulate software ecosystem so that software stays insecure.
- Break into computers; access hundreds of millions of disks, microphones, cameras.
- Add back doors to hardware.

2012 U.S. government report says that Chinese-manufactured routers provide “Chinese intelligence services access to telecommunication networks”.

Some important clarifications

1. “We” doesn’t include me. I want secure crypto.
2. Their actions violate fundamental human rights.
3. I don’t know how much of today’s crypto ecosystem was deliberately manipulated.

This talk is actually a thought experiment: how could an attacker manipulate the ecosystem for insecurity?

Timing attacks


Almost all AES implementations use fast lookup tables. Kernel’s secret AES key influences table-load addresses, influencing CPU cache state, influencing measurable timings of the attack process.

65ms: compute key from timings.
Some important clarifications

1. “We” doesn’t include me.
   I want secure crypto.

2. Their actions violate fundamental human rights.

3. I don’t know how much of today’s crypto ecosystem was deliberately manipulated.

This talk is actually a thought experiment:
how could an attacker manipulate the ecosystem for insecurity?

Timing attacks

Attack process on same CPU but without privileges.

Almost all AES implementations use fast lookup tables.
Kernel’s secret AES key influences table-load addresses,
influencing CPU cache state,
influencing measurable timings of the attack process.
65ms: compute key from timings.
Other useful strategies, not covered in this talk:
Manipulate software ecosystem so that software stays insecure.
Break into computers; access hundreds of millions of disks, screens, microphones, cameras.
Add back doors to hardware.

Some important clarifications
1. “We” doesn’t include me. I want secure crypto.
2. Their actions violate fundamental human rights.
3. I don’t know how much of today’s crypto ecosystem was deliberately manipulated.

This talk is actually a thought experiment: how could an attacker manipulate the ecosystem for insecurity?

Timing attacks
Attack process on same CPU, but without privileges.
Almost all AES implementations use fast lookup tables.
Kernel’s secret AES key influences table-load addresses, influencing CPU cache state, influencing measurable timings of the attack process.
65ms: compute key from timings.
Some important clarifications

1. “We” doesn’t include me. I want secure crypto.
2. Their actions violate fundamental human rights.
3. I don’t know how much of today’s crypto ecosystem was deliberately manipulated.

This talk is actually a thought experiment: how could an attacker manipulate the ecosystem for insecurity?

Timing attacks


Almost all AES implementations use fast lookup tables. Kernel’s secret AES key influences table-load addresses, influencing CPU cache state, influencing measurable timings of the attack process.

65ms: compute key from timings.
Some important clarifications

1. “We” doesn’t include me.
2. Their actions violate fundamental human rights.
3. I don’t know how much today’s crypto ecosystem was deliberately manipulated.
4. This talk is actually a thought experiment: how could an attacker manipulate the ecosystem for insecurity?

Timing attacks


Almost all AES implementations use fast lookup tables. Kernel’s secret AES key influences table-load addresses, influencing CPU cache state, influencing measurable timings of the attack process. 65ms: compute key from timings.

2011 Brumley–Tuveri: minutes to steal another machine’s OpenSSL ECDSA key. Secret branch conditions influence timings.

Most cryptographic software has many small-scale variations: e.g., memcmp for IPsec MACs.

Many more timing attacks: e.g., 2014 van de Pol–Smart–Yarom extracted Bitcoin secret keys from 25 OpenSSL signatures.
Some important clarifications
1. “We” doesn’t include me.
2. Their actions violate fundamental human rights.
3. I don’t know how much of today’s crypto ecosystem was deliberately manipulated.

This talk is actually a thought experiment: how could an attacker manipulate the ecosystem for insecurity?

Timing attacks

Almost all AES implementations use fast lookup tables. Kernel’s secret AES key influences table-load addresses, influencing CPU cache state, influencing measurable timings of the attack process. 65ms: compute key from timings.

2011 Brumley–Tuveri: minutes to steal another machine’s OpenSSL ECDSA key. Secret branch conditions influence timings.

Most cryptographic software has many more small variations in timing: e.g., memcmp for IPsec MACs.

Many more timing attacks: e.g., 2014 van de Pol–Smart–Yarom extracted Bitcoin secret keys from 25 OpenSSL signatures.
Some important clarifications

1. "We" doesn't include me.
I want secure crypto.

2. Their actions violate fundamental human rights.

3. I don't know how much of today's crypto ecosystem was deliberately manipulated.

This talk is actually a thought experiment: how could an attacker manipulate the ecosystem for insecurity?

Timing attacks

Attack process on same CPU but without privileges.

Almost all AES implementations use fast lookup tables.
Kernel’s secret AES key influences table-load addresses, influencing CPU cache state, influencing measurable timings of the attack process.
65ms: compute key from timings.

2011 Brumley–Tuveri: minutes to steal another machine’s OpenSSL ECDSA key.
Secret branch conditions influence timings.

Most cryptographic software has many more small-scale variations in timing: e.g., memcmp for IPsec MAC.

Timing attacks


Almost all AES implementations use fast lookup tables. Kernel’s secret AES key influences table-load addresses, influencing CPU cache state, influencing measurable timings of the attack process. 65ms: compute key from timings.

2011 Brumley–Tuveri: minutes to steal another machine’s OpenSSL ECDSA key. Secret branch conditions influence timings.

Most cryptographic software has many more small-scale variations in timing: e.g.,\s memcmp for IPsec MACs.

Many more timing attacks: e.g. 2014 van de Pol–Smart–Yarom extracted Bitcoin secret keys from 25 OpenSSL signatures.
Timing attacks


Almost all AES implementations use fast lookup tables. Kernel’s secret AES key influences table-load addresses, influencing CPU cache state, influencing measurable timings of the attack process. Compute key from timings.

2011 Brumley–Tuveri: minutes to steal another machine’s OpenSSL ECDSA key. Secret branch conditions influence timings.

Most cryptographic software has many more small-scale variations in timing: e.g., memcmp for IPsec MACs.

Many more timing attacks: e.g. 2014 van de Pol–Smart–Yarom extracted Bitcoin secret keys from 25 OpenSSL signatures.

Manufacture public denials that such attacks exist. Maybe terrorists Alice and Bob won’t try.

2001 NIST “Report on the development of the Advanced Encryption Standard (AES)”: “A general defense against timing attacks is to ensure that each encryption and decryption operation runs in the same amount of time. Table lookup: not vulnerable to timing attacks.”
Timing attacks


Almost all AES implementations use fast lookup tables. Kernel's secret AES key influences table-load addresses, influencing CPU cache state, influencing measurable timings of the attack process.

65ms: compute key from timings.


Most cryptographic software has many more small-scale variations in timing: e.g., memcmp for IPsec MACs.

Many more timing attacks: e.g.,


Manufacture public denials that such attacks exist.

Maybe terrorists Alice and Bob won't try to stop the attacks.


“A general defense against timing attacks is to ensure that each encryption and decryption operation runs in the same amount of time. Table lookup is not vulnerable to timing attacks.
Timing attacks


Almost all AES implementations use fast lookup tables. Kernel's secret AES key influences table-load addresses, influencing CPU cache state, influencing measurable timings of the attack process. 65ms: compute key from timings.

2011 Brumley–Tuveri: minutes to steal another machine’s OpenSSL ECDSA key. Secret branch conditions influence timings.

Most cryptographic software has many more small-scale variations in timing: e.g., memcmp for IPsec MACs.

Many more timing attacks: e.g. 2014 van de Pol–Smart–Yarom extracted Bitcoin secret keys from 25 OpenSSL signatures.

 Manufacture public denials that such attacks exist.

Maybe terrorists Alice and Bob won’t try to stop the attacks.

2001 NIST “Report on the development of the Advanced Encryption Standard (AES)”:

“A general defense against timing attacks is to ensure that each encryption and decryption operation runs in the same amount of time. . . . Table lookup: not vulnerable to timing attacks.”
2011 Brumley–Tuveri: minutes to steal another machine’s OpenSSL ECDSA key. Secret branch conditions influence timings.

Most cryptographic software has many more small-scale variations in timing: e.g., `memcmp` for IPsec MACs.

Many more timing attacks: e.g. 2014 van de Pol–Smart–Yarom extracted Bitcoin secret keys from 25 OpenSSL signatures.

Manufacture public denials that such attacks exist.

Maybe terrorists Alice and Bob won’t try to stop the attacks.

2001 NIST “Report on the development of the Advanced Encryption Standard (AES)”: “A general defense against timing attacks is to ensure that each encryption and decryption operation runs in the same amount of time. . . . Table lookup: not vulnerable to timing attacks.”
Manufacture public denials that such attacks exist.

Maybe terrorists Alice and Bob won’t try to stop the attacks.

2001 NIST “Report on the development of the Advanced Encryption Standard (AES)”: “A general defense against timing attacks is to ensure that each encryption and decryption operation runs in the same amount of time.

Table lookup: not vulnerable to timing attacks.”
Manufacture public denials that such attacks exist.

Maybe terrorists Alice and Bob won't try to stop the attacks.

2001 NIST “Report on the development of the Advanced Encryption Standard (AES)”: “A general defense against timing attacks is to ensure that each encryption and decryption operation runs in the same amount of time. ... Table lookup: not vulnerable to timing attacks.”

2008 RFC 5246 “The Transport Layer Security (TLS) Protocol, Version 1.2”: “This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal.”
Manufacture public denials that such attacks exist.

Maybe terrorists Alice and Bob won’t try to stop the attacks.

2001 NIST “Report on the development of the Advanced Encryption Standard (AES)”: “A general defense against timing attacks is to ensure that each encryption and decryption operation runs in the same amount of time. . . . Table lookup: not vulnerable to timing attacks.”

2008 RFC 5246 “The Transport Layer Security (TLS) Protocol, Version 1.2”: “This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal.”
Manufacture public denials that such attacks exist.

Maybe terrorists Alice and Bob won’t try to stop the attacks.

2001 NIST “Report on the development of the Advanced Encryption Standard (AES)”: “A general defense against timing attacks is to ensure that each encryption and decryption operation runs in the same amount of time. . . . Table lookup: not vulnerable to timing attacks.”

2008 RFC 5246 “The Transport Layer Security (TLS) Protocol, Version 1.2”: “This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal.”
Manufacture public denials that such attacks exist.
Maybe terrorists Alice and Bob won’t try to stop the attacks.

2001 NIST “Report on the development of the Advanced Encryption Standard (AES)”: “A general defense against timing attacks is to ensure that each encryption and decryption operation runs in the same amount of time. ... Table lookup: not vulnerable to timing attacks.”

2008 RFC 5246 “The Transport Layer Security (TLS) Protocol, Version 1.2”: “This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal.”

2013 AlFardan–Paterson “Lucky Thirteen: breaking the TLS and DTLS record protocols”: exploit these timings; steal plaintext.
Manufacture public denials that such attacks exist. Maybe terrorists Alice and Bob won’t try to stop the attacks.

2008 RFC 5246 “The Transport Layer Security (TLS) Protocol, Version 1.2”: “This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal.”

2013 AlFardan–Paterson “Lucky Thirteen: breaking the TLS and DTLS record protocols”: exploit these timings; steal plaintext.

What if Alice and Bob use crypto software built solely from these instructions? Yikes: we won’t see anything from timings!

Some instructions have no data flow from their inputs to CPU timings: e.g., logic instructions, constant-distance shifts, multiply (on most CPUs), add, subtract.
Manufacture public denials that such attacks exist. Maybe terrorists Alice and Bob won't try to stop the attacks.

2001 NIST "Report on the development of the Advanced Encryption Standard (AES)":

"A general defense against timing attacks is to ensure that each encryption and decryption operation runs in the same amount of time. : : : Table lookup: not vulnerable to timing attacks."


"This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal."

2013 AlFardan–Paterson "Lucky Thirteen: breaking the TLS and DTLS record protocols":

"exploit these timings; steal plaintext."

What if Alice and Bob use crypto software built solely from these instructions? Yikes: we won't see anything from timings!

Some instructions have no data flow from their inputs to CPU timings: e.g., constant-distance shifts, multiply (on most CPUs), logic instructions, etc.
Manufacture public denials that such attacks exist. Maybe terrorists Alice and Bob won't try to stop the attacks.

2001 NIST "Report on the development of the Advanced Encryption Standard (AES)": “A general defense against timing attacks is to ensure that each encryption and decryption operation runs in the same amount of time. Table lookup: not vulnerable to timing attacks.”

2008 RFC 5246 "The Transport Layer Security (TLS) Protocol, Version 1.2": “This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal.”

2013 AlFardan–Paterson “Lucky Thirteen: breaking the TLS and DTLS record protocols”: exploit these timings; steal plaintext.

Some instructions have no data flow from their inputs to CPU timings: e.g., logic instructions, constant-distance shifts, multiply (on most CPUs), add, subtract.

What if Alice and Bob use crypto software built solely from these instructions? Yikes: we won't see anything from timings!
2008 RFC 5246 “The Transport Layer Security (TLS) Protocol, Version 1.2”: “This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal.”

2013 AlFardan–Paterson “Lucky Thirteen: breaking the TLS and DTLS record protocols”: exploit these timings; steal plaintext.

Some instructions have no data flow from their inputs to CPU timings: e.g., logic instructions, constant-distance shifts, multiply (on most CPUs), add, subtract.

What if Alice and Bob use crypto software built solely from these instructions? Yikes: we won’t see anything from timings!
2008 RFC 5246 “The Transport Layer Security (TLS) Protocol, Version 1.2”: “This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal.”

2013 AlFardan–Paterson “Lucky Thirteen: breaking the TLS and DTLS record protocols”: exploit these timings; steal plaintext.

Some instructions have no data flow from their inputs to CPU timings: e.g., logic instructions, constant-distance shifts, multiply (on most CPUs), add, subtract.

What if Alice and Bob use crypto software built solely from these instructions? Yikes: we won’t see anything from timings!

Try to scare implementors away from constant-time software.

e.g. “It will be too slow.”

“It’s too hard to write.”
RFC 5246 “The Transport Layer Security (TLS) Protocol, Version 1.2”: “This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal.”

Some instructions have no data flow from their inputs to CPU timings: e.g., logic instructions, constant-distance shifts, multiply (on most CPUs), add, subtract.

What if Alice and Bob use crypto software built solely from these instructions? Yikes: we won’t see anything from timings!

Try to scare implementors away from constant-time software.

Fund variable-time software, maybe with “countermeasures” that make the timings difficult for researchers to analyze but that are still breakable with our computer resources.

Fardan–Paterson “Lucky Thirteen: breaking the TLS and DTLS record protocols”: exploit these timings; steal plaintext.

e.g. “It will be too slow.”
“It’s too hard to write.”
The Transport Layer Security (TLS) Protocol, Version 1.2: “This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be exploitable, due to the large block size of existing MACs and the small size of the timing signal.”

AlFardan–Paterson “Lucky Thirteen: breaking the TLS and DTLS record protocols”: exploit these timings; steal plaintext.

Some instructions have no data flow from their inputs to CPU timings: e.g., logic instructions, constant-distance shifts, multiply (on most CPUs), add, subtract.

What if Alice and Bob use crypto software built solely from these instructions? Yikes: we won’t see anything from timings!

Try to scare implementors away from constant-time software. e.g. “It will be too slow.”
“It’s too hard to write.”

Fund variable-time software, maybe with “countermeasures” that make the timings difficult for researchers to analyze but that are still breakable with our computer resources.
Some instructions have no data flow from their inputs to CPU timings: e.g., logic instructions, constant-distance shifts, multiply (on most CPUs), add, subtract.

What if Alice and Bob use crypto software built solely from these instructions? Yikes: we won’t see anything from timings!

Try to scare implementors away from constant-time software. e.g. “It will be too slow.”
“It’s too hard to write.”

Fund variable-time software, maybe with “countermeasures” that make the timings difficult for researchers to analyze but that are still breakable with our computer resources.
Some instructions have no data flow from their inputs to CPU timings: e.g., logic instructions, constant-distance shifts, multiply (on most CPUs), add, subtract.

What if Alice and Bob use crypto software built solely from these instructions? Yikes: we won’t see anything from timings!

Try to scare implementors away from constant-time software.

e.g. “It will be too slow.”
“It’s too hard to write.”

Fund variable-time software, maybe with “countermeasures” that make the timings difficult for researchers to analyze but that are still breakable with our computer resources.
Some instructions have no data flow from their inputs to CPU timings: e.g., logic instructions, constant-distance shifts, multiply (on most CPUs), add, subtract.

What if Alice and Bob use crypto software built solely from these instructions? Yikes: we won’t see anything from timings!

Try to scare implementors away from constant-time software. e.g. “It will be too slow.” “It’s too hard to write.”

Fund variable-time software, maybe with “countermeasures” that make the timings difficult for researchers to analyze but that are still breakable with our computer resources.

Continue expressing skepticism that constant time is needed. e.g. 2012 Mowery–Keelveedhi–Shacham “Are AES x86 cache timing attacks still feasible?”, unfortunately shredded by 2014 Irazoqui–Inci–Eisenbarth–Sunar “Wait a minute! A fast, cross-VM attack on AES”.
Some instructions have no data flow from their inputs to CPU timings: e.g., logic instructions, constant-distance shifts, multiply (on most CPUs), add, subtract.

What if Alice and Bob use crypto software built solely from these instructions? Yikes: we won’t see anything from timings!

Try to scare implementors away from constant-time software. e.g. “It will be too slow.”

Fund variable-time software, maybe with “countermeasures” that make the timings difficult for researchers to analyze but that are still breakable with our computer resources.

Continue expressing skepticism that constant time is needed. e.g. 2012 Mowery–Keelveedhi–Shacham “Are AES x86 cache timing attacks still feasible?”, unfortunately shredded by 2014 Irazoqui–Inci–Eisenbarth–Sunar “Wait a minute! A fast, cross-VM attack on AES”.

What if terrorists Alice and Bob use a different cipher for which constant-time implementations are simple and fast? Yikes! Don’t standardize that cipher. e.g. choose Rijndael as AES, not higher-security Serpent.

Watch out for any subsequent standardization efforts. Don’t standardize just because it’s new.

Discourage use of the cipher. Pretend that standardization is a guarantee of security while anything non-standard has questionable security.
Some instructions have no data flow from their inputs to CPU timings: e.g., logic instructions, constant-distance shifts, multiply, add, subtract.

What if Alice and Bob use crypto software built solely from these instructions? Yikes: we won’t see anything from timings!

Try to scare implementors away from constant-time software.

e.g. “It will be too slow.”

“ It’s too hard to write.”

Fund variable-time software, maybe with “countermeasures” that make the timings difficult for researchers to analyze but that are still breakable with our computer resources.

Continue expressing skepticism that constant time is needed.

e.g. 2012 Mowery–Keelveedhi–Shacham “Are AES x86 cache timing attacks still feasible?”, unfortunately shredded by 2014 Irazoqui–Inci–Eisenbarth–Sunar “Wait a minute! A fast, cross-VM attack on AES”.

What if terrorists Alice and Bob use a different cipher for which constant-time implementations are simple and fast?

Don’t standardize that cipher.

e.g. choose Rijndael as AES, not higher-security Serpent.

Watch out for any subsequent standardization efforts.

Discourage use of the cipher.

Pretend that standardization is a guarantee of security while anything non-standard has questionable security.
Fund variable-time software, maybe with “countermeasures” that make the timings difficult for researchers to analyze but that are still breakable with our computer resources. Continue expressing skepticism that constant time is needed. e.g. 2012 Mowery–Keelveedhi–Shacham “Are AES x86 cache timing attacks still feasible?”, unfortunately shredded by 2014 Irazoqui–Inci–Eisenbarth–Sunar “Wait a minute! A fast, cross-VM attack on AES”.

What if terrorists Alice and Bob use a different cipher for which constant-time implementations are simple and fast? Yikes! Don’t standardize that cipher, e.g. choose Rijndael as AES, not higher-security Serpent. Watch out for any subsequent standardization efforts.

Discourage use of the cipher. Pretend that standardization is a guarantee of security while anything non-standard has questionable security.
Fund variable-time software, maybe with “countermeasures” that make the timings difficult for researchers to analyze but that are still breakable with our computer resources.

Continue expressing skepticism that constant time is needed. e.g. 2012 Mowery–Keelveedhi–Shacham “Are AES x86 cache timing attacks still feasible?”, unfortunately shredded by 2014 Irazoqui–Inci–Eisenbarth–Sunar “Wait a minute! A fast, cross-VM attack on AES”.

What if terrorists Alice and Bob use a different cipher for which constant-time implementations are simple and fast? Yikes!

Don’t standardize that cipher. e.g. choose Rijndael as AES, not higher-security Serpent. Watch out for any subsequent standardization efforts.

Discourage use of the cipher. Pretend that standardization is a guarantee of security while anything non-standard has questionable security.
Variable-time software, maybe with “countermeasures” that make the timings difficult for researchers to analyze but that are still breakable by computer resources.

Are expressing skepticism that constant time is needed.


What if terrorists Alice and Bob use a different cipher for which constant-time implementations are simple and fast? Yikes!

Don’t standardize that cipher. e.g. choose Rijndael as AES, not higher-security Serpent. Watch out for any subsequent standardization efforts.

Discourage use of the cipher. Pretend that standardization is a guarantee of security while anything non-standard has questionable security.

1998 Bleichenbacher: Decrypt SSL RSA ciphertext by observing server responses to $\approx 10^6$ variants of ciphertext.

SSL first inverts RSA, then checks for “PKCS padding” (which many forgeries have). Subsequent processing applies more serious integrity checks.

Server responses reveal pattern of PKCS forgeries; pattern reveals plaintext.
Fund variable-time software, maybe with "countermeasures" that make the timings difficult for researchers to analyze but that are still breakable with our computer resources. Continue expressing skepticism that constant time is needed. E.g. 2012 Mowery–Keelveedhi–Shacham "Are AES x86 cache timing attacks still feasible?", unfortunately shredded by 2014 Irazoqui–Inci–Eisenbarth–Sunar "Wait a minute! A fast, cross-VM attack on AES".

What if terrorists Alice and Bob use a different cipher for which constant-time implementations are simple and fast? Yikes! Don't standardize that cipher. E.g. choose Rijndael as AES, not higher-security Serpent. Watch out for any subsequent standardization efforts. Discourage use of the cipher. Pretend that standardization is a guarantee of security while anything non-standard has questionable security.

Padding oracles
1998 Bleichenbacher: Decrypt SSL RSA ciphertext by observing server responses to $\approx 10^6$ variants of ciphertext. SSL first inverts RSA, then checks for "PKCS padding" (which many forgeries have). Subsequent processing applies more serious integrity checks. Server responses reveal pattern of PKCS forgeries; pattern reveals plaintext.
Fund variable-time software, maybe with “countermeasures” that make the timings difficult for researchers to analyze but that are still breakable with our computer resources.

Continue expressing skepticism that constant time is needed. e.g. 2012 Mowery–Keelveedhi–Shacham “Are AES x86 cache timing attacks still feasible?”, unfortunately shredded by 2014 Irazoqui–Inci–Eisenbarth–Sunar “Wait a minute! A fast, cross-VM attack on AES”.

What if terrorists Alice and Bob use a different cipher for which constant-time implementations are simple and fast? Yikes! Don’t standardize that cipher. e.g. choose Rijndael as AES, not higher-security Serpent. Watch out for any subsequent standardization efforts.

Discourage use of the cipher. Pretend that standardization is a guarantee of security while anything non-standard has questionable security.

Padding oracles
1998 Bleichenbacher: Decrypt SSL RSA ciphertext by observing server responses to ≈10⁶ variants of ciphertext.

SSL first inverts RSA, then checks for “PKCS padding” (which many forgeries have). Subsequent processing applies more serious integrity checks.

Server responses reveal pattern of PKCS forgeries; pattern reveals plaintext.
What if terrorists Alice and Bob use a different cipher for which constant-time implementations are simple and fast? Yikes!

Don’t standardize that cipher. e.g. choose Rijndael as AES, not higher-security Serpent. Watch out for any subsequent standardization efforts.

Discourage use of the cipher. Pretend that standardization is a guarantee of security while anything non-standard has questionable security.

Padding oracles
1998 Bleichenbacher: Decrypt SSL RSA ciphertext by observing server responses to \(\approx 10^6\) variants of ciphertext.

SSL first inverts RSA, then checks for “PKCS padding” (which many forgeries have). Subsequent processing applies more serious integrity checks.

Server responses reveal pattern of PKCS forgeries; pattern reveals plaintext.
What if terrorists Alice and Bob use a different cipher for which constant-time implementations are simple and fast? Yikes! Don’t standardize that cipher. Choose Rijndael as AES, not higher-security Serpent. Watch out for any subsequent standardization efforts.

Forage use of the cipher. Pretend that standardization is a guarantee of security while anything non-standard has questionable security.

Padding oracles

1998 Bleichenbacher: Decrypt SSL RSA ciphertext by observing server responses to $\approx 10^6$ variants of ciphertext.

SSL first inverts RSA, then checks for “PKCS padding” (which many forgeries have). Subsequent processing applies more serious integrity checks.

Server responses reveal pattern of PKCS forgeries; pattern reveals plaintext.

Design cryptographic systems so that forgeries are sent through as much processing as possible.

e.g. Design SSL to decrypt and check padding before checking a serious MAC. Broken by padding-oracle attacks such as BEAST and POODLE.

What if terrorists Alice and Bob use a different cipher for which constant-time implementations are simple and fast? Yikes! Don't standardize that cipher. e.g. choose Rijndael as AES, not higher-security Serpent. Watch out for any subsequent standardization efforts.

Discourage use of the cipher. Pretend that standardization is a guarantee of security while anything non-standard has questionable security.

---

**Padding oracles**

1998 Bleichenbacher:
Decrypt SSL RSA ciphertext by observing server responses to $\approx 10^6$ variants of ciphertext.

SSL first inverts RSA, then checks for “PKCS padding” (which many forgeries have). Subsequent processing applies more serious integrity checks.

Server responses reveal pattern of PKCS forgeries; pattern reveals plaintext.

---

Design cryptographic systems so that forgeries are sent through as much processing as possible.

e.g. Design SSL to decrypt and check padding before checking a serious MAC. Broken by padding-oracle attacks such as BEAST and POODLE.

e.g. Design “encrypt-only” IPsec options. Broken by 2006 Paterson–Yau for Linux and 2007 Degabriele–Paterson for RFCs.
What if terrorists Alice and Bob use a different cipher for which constant-time implementations are simple and fast? Yikes! Don't standardize that cipher. e.g. choose Rijndael as AES, not higher-security Serpent. Watch out for any subsequent standardization efforts. Discourage use of the cipher. Pretend that standardization is a guarantee of security while anything non-standard has questionable security.

Padding oracles
1998 Bleichenbacher: Decrypt SSL RSA ciphertext by observing server responses to $\approx 10^6$ variants of ciphertext. SSL first inverts RSA, then checks for “PKCS padding” (which many forgeries have). Subsequent processing applies more serious integrity checks. Server responses reveal pattern of PKCS forgeries; pattern reveals plaintext.

Design cryptographic systems so that forgeries are sent through as much processing as possible. e.g. Design SSL to decrypt and check padding before checking a serious MAC. Broken by padding-oracle attacks such as BEAST and POODLE. e.g. Design “encrypt-only” IPsec options. Broken by 2006 Paterson–Yau for Linux and 2007 Degabriele–Paterson for RFCs.
Padding oracles

1998 Bleichenbacher:
Decrypt SSL RSA ciphertext by observing server responses to $\approx 10^6$ variants of ciphertext.

SSL first inverts RSA, then checks for "PKCS padding" (which many forgeries have). Subsequent processing applies more serious integrity checks.

Server responses reveal pattern of PKCS forgeries; pattern reveals plaintext.

Design cryptographic systems so that forgeries are sent through as much processing as possible.

e.g. Design SSL to decrypt and check padding before checking a serious MAC. Broken by padding-oracle attacks such as BEAST and POODLE.

e.g. Design "encrypt-only" IPsec options. Broken by 2006 Paterson–Yau for Linux and 2007 Degabriele–Paterson for RFCs.
Design cryptographic systems so that forgeries are sent through as much processing as possible.

e.g. Design SSL to decrypt and check padding before checking a serious MAC. Broken by padding-oracle attacks such as BEAST and POODLE.

e.g. Design “encrypt-only” IPsec options. Broken by 2006 Paterson–Yau for Linux and 2007 Degabriele–Paterson for RFCs.

Randomness

1995 Goldberg–Wagner: Netscape SSL keys had < 50 bits of entropy.

2008 Bello: Debian/Ubuntu OpenSSL keys for years had < 20 bits of entropy.

2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and 2012 Heninger–Durumeric–Wustrow–Halderman broke the RSA public keys for 0.5% of all SSL servers.

The primes had so little randomness that they collided.
Design cryptographic systems so that forgeries are sent through as much processing as possible.

e.g. Design SSL to decrypt and check padding before checking a serious MAC. Broken by padding-oracle attacks such as BEAST and POODLE.

1998 Bleichenbacher: Decrypt SSL RSA ciphertext by observing server responses to $\approx 10^6$ variants of ciphertext.

SSL first inverts RSA, then checks for "PKCS padding" (which many forgeries have). Subsequent processing applies more serious integrity checks. Server responses reveal pattern of PKCS forgeries; pattern reveals plaintext.

1995 Goldberg–Wagner: Netscape SSL keys had < 50 bits of entropy.

2008 Bello: Debian/Ubuntu OpenSSL keys for years had < 20 bits of entropy.

2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and 2012 Heninger–Durumeric–Wustrow–Halderman broke the RSA public keys for 0.5% of all SSL servers. The primes had so little randomness that they collided.
Design cryptographic systems so that forgeries are sent through as much processing as possible.

e.g. Design SSL to decrypt and check padding before checking a serious MAC. Broken by padding-oracle attacks such as BEAST and POODLE.

e.g. Design “encrypt-only” IPsec options. Broken by 2006 Paterson–Yau for Linux and 2007 Degabriele–Paterson for RFCs.

Randomness

1995 Goldberg–Wagner: Netscape SSL keys had <50 bits of entropy.

2008 Bello: Debian/Ubuntu OpenSSL keys for years had <20 bits of entropy.

2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and 2012 Heninger–Durumeric–Wustrow–Halderman broke the RSA public keys for 0.5% of all SSL servers. The primes had so little randomness that they collided.
Design cryptographic systems so that forgeries are sent through as much processing as possible.

e.g. Design SSL to decrypt and check padding before checking a serious MAC. Broken by padding-oracle attacks such as BEAST and POODLE.


Randomness

1995 Goldberg–Wagner: Netscape SSL keys had <50 bits of entropy.

2008 Bello: Debian/Ubuntu OpenSSL keys for years had <20 bits of entropy.

2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and 2012 Heninger–Durumeric–Wustrow–Halderman broke the RSA public keys for 0.5% of all SSL servers. The primes had so little randomness that they collided.
Design cryptographic systems so that forgeries are sent through as much processing as possible.

e.g. Design SSL to decrypt and check padding before checking a serious MAC. Broken by padding-oracle attacks such as BEAST and POODLE.


Randomness

1995 Goldberg–Wagner: Netscape SSL keys had <50 bits of entropy.

2008 Bello: Debian/Ubuntu OpenSSL keys for years had <20 bits of entropy.

2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wächter and 2012 Heninger–Durumeric–Wustrow–Halderman broke the RSA public keys for 0.5% of all SSL servers. The primes had so little randomness that they collided.

Make randomness-generation code extremely difficult to audit.

Have each application maintain its own RNG “for speed”.

Maintain separate RNG code for each application and build this RNG from the inputs conveniently available to that application.

Pay people to use backdoored RNGs such as Dual EC.

Claim “provable security”.

Design cryptographic systems so that forgeries are sent through as much processing as possible. e.g. Design SSL to decrypt and check padding before checking a serious MAC. Broken by padding-oracle attacks such as BEAST and POODLE. e.g. Design "encrypt-only" IPsec options. Broken by 2006 Paterson–Yau for Linux and 2007 Degabriele–Paterson for RFCs.

Randomness
1995 Goldberg–Wagner: Netscape SSL keys had <50 bits of entropy.
2008 Bello: Debian/Ubuntu OpenSSL keys for years had <20 bits of entropy.
2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and 2012 Heninger–Durumeric–Wustrow–Halderman broke the RSA public keys for 0.5% of all SSL servers. The primes had so little randomness that they collided.

Make randomness-generation code extremely difficult to audit. Have each application maintain its own RNG “for speed”. Maintain separate RNG code for each application. “For simplicity” build this RNG in ad-hoc ways from the inputs conveniently available to that application. Pay people to use backdoored RNGs such as Dual EC. Claim “provable security”.
<table>
<thead>
<tr>
<th>Randomness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995 Goldberg–Wagner: Netscape SSL keys had &lt;50 bits of entropy.</td>
</tr>
<tr>
<td>2008 Bello: Debian/Ubuntu OpenSSL keys for years had &lt;20 bits of entropy.</td>
</tr>
<tr>
<td>2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and 2012 Heninger–Durumeric–Wustrow–Halderman broke the RSA public keys for 0.5% of all SSL servers. The primes had so little randomness that they collided.</td>
</tr>
</tbody>
</table>

Make randomness-generation code extremely difficult to audit. Have each application maintain its own RNG “for speed”. Maintain separate RNG code for each application. “For simplicity” build this RNG in ad-hoc ways from the inputs conveniently available to that application. Pay people to use backdoored RNGs such as Dual EC. Claim “provable security”. 
Randomness

1995 Goldberg–Wagner: Netscape SSL keys had < 50 bits of entropy.

2008 Bello: Debian/Ubuntu OpenSSL keys for years had < 20 bits of entropy.

2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and 2012 Heninger–Durumeric–Wustrow–Halderman broke the RSA public keys for 0.5% of all SSL servers. The primes had so little randomness that they collided.

Make randomness-generation code extremely difficult to audit.

Have each application maintain its own RNG “for speed”.

Maintain separate RNG code for each application. “For simplicity” build this RNG in ad-hoc ways from the inputs conveniently available to that application.

Pay people to use backdoored RNGs such as Dual EC.

Claim “provable security”.
Randomness

1995 Goldberg–Wagner: Netscape SSL keys had <50 bits of entropy.

2008 Bello: Debian/Ubuntu OpenSSL keys for years had <20 bits of entropy.

2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and 2012 Heninger–Durumeric–Wustrow–Halderman broke the RSA public keys for 0.5% of all SSL servers.

The primes had so little randomness that they collided.

Make randomness-generation code extremely difficult to audit.

Have each application maintain its own RNG “for speed”.

Maintain separate RNG code for each application. “For simplicity” build this RNG in ad-hoc ways from the inputs conveniently available to that application.

Pay people to use backdoored RNGs such as Dual EC.

Claim “provable security”.

What if the terrorists merge all available inputs into a central entropy pool?

This pool can survive many bad/failing/malicious inputs if there is one good input.

Merging process is auditable.

Yikes!
1995 Goldberg–Wagner: Netscape SSL keys had < 50 bits of entropy.

2008 Bello: Debian/Ubuntu OpenSSL keys for years had < 20 bits of entropy.

2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and 2012 Heninger–Durumeric–Wustrow–Halderman broke the RSA public keys for 0.5% of all SSL servers. The primes had so little randomness that they collided.

Make randomness-generation code extremely difficult to audit.

Have each application maintain its own RNG “for speed”.

Maintain separate RNG code for each application. “For simplicity” build this RNG in ad-hoc ways from the inputs conveniently available to that application.

Pay people to use backdoored RNGs such as Dual EC.

What if the terrorists merge all available inputs into a central entropy pool?

This pool can survive many bad/failing/malicious inputs, if there is one good input. Merging process is auditable. Yikes!
Randomness

1995 Goldberg–Wagner: Netscape SSL keys had < 50 bits of entropy.

2008 Bello: Debian/Ubuntu OpenSSL keys for years had < 20 bits of entropy.

2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and 2012 Heninger–Durumeric–Wustrow–Halderman broke the RSA public keys for 0.5% of all SSL servers. The primes had so little randomness that they collided.

Make randomness-generation code extremely difficult to audit.

Have each application maintain its own RNG “for speed”.

Maintain separate RNG code for each application. “For simplicity” build this RNG in ad-hoc ways from the inputs conveniently available to that application.

Pay people to use backdoored RNGs such as Dual EC. Claim “provable security”.

What if the terrorists merge all available inputs into a central entropy pool?

This pool can survive many bad/failing/malicious inputs if there is one good input. Merging process is auditable. Yikes!
Make randomness-generation code extremely difficult to audit.

Have each application maintain its own RNG “for speed”.

Maintain separate RNG code for each application. “For simplicity” build this RNG in ad-hoc ways from the inputs conveniently available to that application.

Pay people to use backdoored RNGs such as Dual EC. Claim “provable security”.

What if the terrorists merge all available inputs into a central entropy pool?

This pool can survive many bad/failing/malicious inputs if there is one good input. Merging process is auditable. Yikes!
Make randomness-generation code extremely difficult to audit.

Have each application maintain its own RNG “for speed”.

Maintain separate RNG code for each application. “For simplicity” build this RNG in ad-hoc ways from the inputs conveniently available to that application.

Pay people to use backdoored RNGs such as Dual EC. Claim “provable security”.

What if the terrorists merge all available inputs into a central entropy pool?

This pool can survive many bad/failing/malicious inputs if there is one good input. Merging process is auditable. Yikes!

Claim performance problems in writing to a central pool, reading from a central pool. Modify pool to make it unusable (random) or scary (urandom).
Randomness-generation code should be extremely difficult to audit.

Each application should maintain its own RNG “for speed”.

Separate RNG code for each application. “For simplicity” build this RNG in ad-hoc ways using inputs conveniently available to that application.

Pay people to use backdoored RNGs such as Dual EC.

Claim “provable security”.

What if the terrorists merge all available inputs into a central entropy pool?

This pool can survive many bad/failing/malicious inputs if there is one good input. Merging process is auditable.

Yikes!

Claim performance problems in writing to a central pool, reading from a central pool.

Modify pool to make it unusable (random) or scary (urandom).

What if the terrorists realize that RNG speed isn’t an issue?

Make it an issue! Design crypto to use randomness as often as possible. This also complicates tests, encouraging bugs.

e.g. DSA use a new random $k$ to sign $m$; could have replaced $k$ with $H(s, m)$. 1992 Rivest: “the poor user is given enough rope with which to hang himself”. 2010 Bushings--Marcan--Segher--Sven “PS3 epic fail”: PS3 forgeries.
What if the terrorists merge all available inputs into a central entropy pool? This pool can survive many bad/failing/malicious inputs if there is one good input. Merging process is auditable.

Yikes!

Claim performance problems in writing to a central pool, reading from a central pool. Modify pool to make it unusable (random) or scary (urandom).

What if the terrorists realize that RNG speed isn’t an issue? Make it an issue! Design crypto to use randomness as often as possible. This also complicates tests, encouraging bugs.

e.g. DSA and ECDSA use a new random number \(k\) to sign \(m\); could have replaced \(k\) with \(H(s, m)\). 1992 Rivest: “the poor user is given enough rope with which to hang himself”. 2010 Bushing–Marcan–Segher–Sven “PS3 epic fail”: PS3 forgeries.
What if the terrorists merge all available inputs into a central entropy pool?

This pool can survive many bad/failing/malicious inputs if there is one good input.
Merging process is auditable.

Yikes!

Claim performance problems in writing to a central pool, reading from a central pool.
Modify pool to make it unusable (random) or scary (urandom).

What if the terrorists realize that RNG speed isn’t an issue?

Make it an issue! Design crypto to use randomness as often as possible. This also complicates tests, encouraging bugs.

E.g. DSA and ECDSA use a new random number k to sign m; could have replaced k with \( H(s, m) \). 1992 Rivest: “the poor user is given enough rope with which to hang himself”. 2010 Bushing–Marcan–Segher–Sven “PS3 epic fail”: PS3 forgeries.
What if the terrorists merge all available inputs into a central entropy pool?

This pool can survive many bad/failing/malicious inputs if there is one good input. Merging process is auditable. Yikes!

Claim performance problems in writing to a central pool, reading from a central pool. Modify pool to make it unusable (random) or scary (urandom).

What if the terrorists realize that RNG speed isn’t an issue?

Make it an issue! Design crypto to use randomness as often as possible. This also complicates tests, encouraging bugs.

e.g. DSA and ECDSA use a new random number $k$ to sign $m$; could have replaced $k$ with $H(s, m)$. 1992 Rivest: “the poor user is given enough rope with which to hang himself”. 2010 Bushing–Marcan–Segher–Sven “PS3 epic fail”: PS3 forgeries.
What if the terrorists merge all available inputs into a central entropy pool? This pool can survive many bad/failing/malicious inputs if there is one good input. The merging process is auditable.

Performance problems in writing to a central pool, reading from a central pool. Modify pool to make it unusable (random) or scary (urandom).

What if the terrorists realize that RNG speed isn’t an issue?

Make it an issue! Design crypto to use randomness as often as possible. This also complicates tests, encouraging bugs.

e.g. DSA and ECDSA use a new random number $k$ to sign $m$; could have replaced $k$ with $H(s, m)$. 1992 Rivest: “the poor user is given enough rope with which to hang himself”. 2010 Bushing–Marcan–Segher–Sven “PS3 epic fail”: PS3 forgeries.

Pure crypto failures

2008 Stevens–Sotirov–Appelbaum–Lenstra–Molnar–Osvik–de Weger exploited MD5 $\Rightarrow$ rogue CA for TLS.
What if the terrorists realize that RNG speed isn’t an issue?

Make it an issue! Design crypto to use randomness as often as possible. This also complicates tests, encouraging bugs.

E.g. DSA and ECDSA use a new random number $k$ to sign $m$; could have replaced $k$ with $H(s, m)$. 1992 Rivest: “the poor user is given enough rope with which to hang himself”. 2010 Bushing–Marcan–Segher–Sven “PS3 epic fail”: PS3 forgeries.

Pure crypto failures

2008 Stevens–Sotirov–Appelbaum–Lenstra–Molnar–Osvik–de Weger exploited MD5 ⇒ rogue CA for TLS.
What if the terrorists realize that RNG speed isn’t an issue?

Make it an issue! Design crypto to use randomness as often as possible. This also complicates tests, encouraging bugs.

e.g. DSA and ECDSA use a new random number $k$ to sign $m$; could have replaced $k$ with $H(s, m)$. 1992 Rivest: “the poor user is given enough rope with which to hang himself”. 2010 Bushing–Marcan–Segher–Sven “PS3 epic fail”: PS3 forgeries.

Pure crypto failures

2008 Stevens–Sotirov–Appelbaum–Lenstra–Molnar–Osvik–de Weger exploited MD5 $\Rightarrow$ rogue CA for TLS.
What if the terrorists realize that RNG speed isn’t an issue?

Make it an issue! Design crypto to use randomness as often as possible. This also complicates tests, encouraging bugs.

e.g. DSA and ECDSA use a new random number $k$ to sign $m$; could have replaced $k$ with $H(s, m)$. 1992 Rivest: “the poor user is given enough rope with which to hang himself”. 2010 Bushing–Marcan–Segher–Sven “PS3 epic fail”: PS3 forgeries.

Pure crypto failures
2008 Stevens–Sotirov–Appelbaum–Lenstra–Molnar–Osvik–de Weger exploited MD5 ⇒ rogue CA for TLS.
What if the terrorists realize that RNG speed isn’t an issue?

Make it an issue! Design crypto to use randomness as often as possible. This also complicates tests, encouraging bugs.

e.g. DSA and ECDSA use a new random number $k$ to sign $m$; could have replaced $k$ with $H(s, m)$. 1992 Rivest: “the poor user is given enough rope with which to hang himself”. 2010 Bushing–Marcan–Segher–Sven “PS3 epic fail” : PS3 forgeries.

Pure crypto failures
2008 Stevens–Sotirov–Appelbaum–Lenstra–Molnar–Osvik–de Weger exploited MD5 ⇒ rogue CA for TLS.
2012 Flame: new MD5 attack.
What if the terrorists realize that RNG speed isn’t an issue?

Make it an issue! Design crypto to use randomness as often as possible. This also complicates tests, encouraging bugs.

e.g. DSA and ECDSA use a new random number $k$ to sign $m$; could have replaced $k$ with $H(s,m)$. 1992 Rivest: “the poor user is given enough rope with which to hang himself”. 2010 Bushing–Marcan–Segher–Sven “PS3 epic fail”: PS3 forgeries.

Pure crypto failures

2008 Stevens–Sotirov–Appelbaum–Lenstra–Molnar–Osvik–de Weger exploited MD5 ⇒ rogue CA for TLS.

2012 Flame: new MD5 attack.

Fact: By 1996, a few years after the introduction of MD5, Preneel and Dobbertin were calling for MD5 to be scrapped.
What if the terrorists realize that RNG speed isn’t an issue?

Make it an issue! Design crypto to use randomness as often as possible. This also complicates tests, encouraging bugs.

e.g. DSA and ECDSA use a new random number $k$ to sign $m$; could have replaced $k$ with $H(s, m)$. 1992 Rivest: “the poor user is given enough rope with which to hang himself”. 2010 Bushing–Marcan–Segher–Sven “PS3 epic fail”: PS3 forgeries.

Pure crypto failures

2008 Stevens–Sotirov–Appelbaum–Lenstra–Molnar–Osvik–de Weger exploited MD5 ⇒ rogue CA for TLS.

2012 Flame: new MD5 attack.

Fact: By 1996, a few years after the introduction of MD5, Preneel and Dobbertin were calling for MD5 to be scrapped.

We managed to keep MD5. How? Speed; standards; compatibility.
What if the terrorists realize that RNG speed isn’t an issue?

Make it an issue! Design crypto to use randomness as often as possible. This also complicates tests, encouraging bugs.

DSA and ECDSA use a random number $k$ to sign $m$ and have replaced $k$ with $H(s; m)$. 1992 Rivest: “the poor user is given enough rope with which to hang himself”. 2010 Bushing–Marcan–Segher–Sven “PS3 epic fail”: PS3 forgeries.

Pure crypto failures

2008 Stevens–Sotirov–Appelbaum–Lenstra–Molnar–Osvik–de Weger exploited MD5 $\Rightarrow$ rogue CA for TLS.

2012 Flame: new MD5 attack.

Fact: By 1996, a few years after the introduction of MD5, Preneel and Dobbertin were calling for MD5 to be scrapped.

We managed to keep MD5. How? Speed; standards; compatibility.

2014: DNSSEC uses RSA-1024 to “secure” IP addresses. e.g. dnssec-deployment.org address is signed by RSA-1024.
What if the terrorists realize that RNG speed isn’t an issue? Make it an issue! Design crypto to use randomness as often as possible. This also complicates tests, encouraging bugs.

For example, DSA uses a new random number $k$ to sign $m$; could have replaced $k$ with $H(s; m)$. 1992 Rivest: “the poor user is given enough rope with which to hang himself”. 2010 Bushing–Marcan–Segher–Sven “PS3 epic fail”: PS3 forgeries.

**Pure crypto failures**

- 2008 Stevens–Sotirov–Appelbaum–Lenstra–Molnar–Osvik–de Weger exploited MD5 $\Rightarrow$ rogue CA for TLS.
- 2012 Flame: new MD5 attack.

Fact: By 1996, a few years after the introduction of MD5, Preneel and Dobbertin were calling for MD5 to be scrapped. We managed to keep MD5. How? Speed; standards; compatibility.

2014: DNSSEC uses RSA-1024 to “secure” IP addresses. e.g. dnssec-deployment.org address is signed by RSA-1024.
What if the terrorists realize that RNG speed isn’t an issue?

Make it an issue! Design crypto to use randomness as often as possible. This also complicates tests, encouraging bugs.

e.g. DSA and ECDSA use a new random number $k$ to sign $m$; could have replaced $k$ with $H(s; m)$. 1992 Rivest: “the poor user is given enough rope with which to hang himself”. 2010 Bushing–Marcan–Segher–Sven “PS3 epic fail”: PS3 forgeries.

Pure crypto failures

2008 Stevens–Sotirov–Appelbaum–Lenstra–Molnar–Osvik–de Weger exploited MD5 $\Rightarrow$ rogue CA for TLS.

2012 Flame: new MD5 attack.

Fact: By 1996, a few years after the introduction of MD5, Preneel and Dobbertin were calling for MD5 to be scrapped.

We managed to keep MD5. How?

Speed; standards; compatibility.

2014: DNSSEC uses RSA-1024 to “secure” IP addresses.
e.g. dnssec-deployment.org address is signed by RSA-1024.
Pure crypto failures

2008 Stevens–Sotirov–Appelbaum–Lenstra–Molnar–Osvik–de Weger exploited MD5 ⇒ rogue CA for TLS.

2012 Flame: new MD5 attack.

Fact: By 1996, a few years after the introduction of MD5, Preneel and Dobbertin were calling for MD5 to be scrapped.

We managed to keep MD5. How? Speed; standards; compatibility.

2014: DNSSEC uses RSA-1024 to “secure” IP addresses. e.g. dnssec-deployment.org address is signed by RSA-1024.
Pure crypto failures
2008 Stevens–Sotirov–
Appelbaum–Lenstra–Molnar–
Osvik–de Weger exploited
MD5 ⇒ rogue CA for TLS.
2012 Flame: new MD5 attack.
Fact: By 1996, a few years
after the introduction of MD5,
Preneel and Dodbertin were
calling for MD5 to be scrapped.
We managed to keep MD5. How?
Speed; standards; compatibility.

2014: DNSSEC uses RSA-1024
to “secure” IP addresses.
e.g. dnssec-deployment.org
address is signed by RSA-1024.
Fact: Analyses in 2003 concluded
that RSA-1024 was breakable;
e.g., 2003 Shamir–Tromer
estimated 1 year, ≈10^7 USD.
Pure crypto failures

2008 Stevens–Sotirov–Appelbaum–Lenstra–Molnar–Osvik–de Weger exploited MD5 ⇒ rogue CA for TLS.

2012 Flame: new MD5 attack.

Fact: By 1996, a few years after the introduction of MD5, Preneel and Dobbertin were calling for MD5 to be scrapped.

We managed to keep MD5. How? Speed; standards; compatibility.

2014: DNSSEC uses RSA-1024 to “secure” IP addresses. e.g. dnssec-deployment.org address is signed by RSA-1024.

Fact: Analyses in 2003 concluded that RSA-1024 was breakable; e.g., 2003 Shamir–Tromer estimated 1 year, $\approx 10^7$ USD.

DNSSEC’s main excuse for sticking to RSA-1024: speed. “Tradeoff between the risk of key compromise and performance.”
Crypto failures

2008 Stevens–Sotirov–
Appelbaum–Lenstra–Molnar–
Osvik–de Weger exploited
MD5 ⇒ rogue CA for TLS.

Name: new MD5 attack.

By 1996, a few years
after the introduction of MD5,
Preneel and Dobbertin were
calling for MD5 to be scrapped.

We managed to keep MD5. How?
Speed; standards; compatibility.

2014: DNSSEC uses RSA-1024
to “secure” IP addresses.
e.g. dnssec-deployment.org
address is signed by RSA-1024.

Fact: Analyses in 2003 concluded
that RSA-1024 was breakable;
e.g., 2003 Shamir–Tromer
estimated 1 year, \( \approx 10^7 \) USD.

DNSSEC’s main excuse
for sticking to RSA-1024: speed.
“Tradeoff between the risk of key
compromise and performance.”

How to convince terrorists
that secure crypto is too slow?
Many techniques: obsolete data,
incompetent benchmarks, fraud.
2008: Stevens–Sotirov–Appelbaum–Lenstra–Molnar–Osvik–de Weger exploited MD5 ⇒ rogue CA for TLS.

2012 Flame: new MD5 attack.

Fact: By 1996, a few years after the introduction of MD5, Preneel and Dobbertin were calling for MD5 to be scrapped. We managed to keep MD5. How? Speed; standards; compatibility.

2014: DNSSEC uses RSA-1024 to “secure” IP addresses.

e.g. dnssec-deployment.org address is signed by RSA-1024.

Fact: Analyses in 2003 concluded that RSA-1024 was breakable; e.g., 2003 Shamir–Tromer estimated 1 year, $\approx 10^7$ USD.

DNSSEC’s main excuse for sticking to RSA-1024: speed. “Tradeoff between the risk of key compromise and performance.”

How to convince terrorists that secure crypto is too slow?

Many techniques: obsolete data, incompetent benchmarks, fraud.
2014: DNSSEC uses RSA-1024 to “secure” IP addresses. e.g. dnssec-deployment.org address is signed by RSA-1024.

Fact: Analyses in 2003 concluded that RSA-1024 was breakable; e.g., 2003 Shamir–Tromer estimated 1 year, \(\approx 10^7\) USD.

DNSSEC’s main excuse for sticking to RSA-1024: speed. “Tradeoff between the risk of key compromise and performance.”

How to convince terrorists that secure crypto is too slow?

Many techniques: obsolete data, incompetent benchmarks, fraud.
2014: DNSSEC uses RSA-1024 to “secure” IP addresses. 
e.g. dnssec-deployment.org address is signed by RSA-1024.

Fact: Analyses in 2003 concluded that RSA-1024 was breakable; 
e.g., 2003 Shamir–Tromer estimated 1 year, $\approx 10^7$ USD.

DNSSEC’s main excuse for sticking to RSA-1024: speed. 
“Tradeoff between the risk of key compromise and performance.”

How to convince terrorists that secure crypto is too slow? 
Many techniques: obsolete data, incompetent benchmarks, fraud.
2014: DNSSEC uses RSA-1024 to “secure” IP addresses.
e.g. dnssec-deployment.org address is signed by RSA-1024.

Fact: Analyses in 2003 concluded that RSA-1024 was breakable;
e.g., 2003 Shamir–Tromer estimated 1 year, \( \approx 10^7 \) USD.

DNSSEC’s main excuse for sticking to RSA-1024: speed.
“Tradeoff between the risk of key compromise and performance.”

How to convince terrorists that secure crypto is too slow?
Many techniques: obsolete data, incompetent benchmarks, fraud.
Example:
“PRESERVE contributes to the security and privacy of future vehicle-to-vehicle and vehicle-to-infrastructure communication systems by addressing critical issues like performance, scalability, and deployability of V2X security systems.”

preserve-project.eu
DNSSEC uses RSA-1024 to "secure" IP addresses. For example, the address `dnssec-deployment.org` is signed by RSA-1024.

Analyses in 2003 concluded that RSA-1024 was breakable; for example, 2003 Shamir–Tromer estimated 1 year, $\approx 10^7$ USD. Thus, RSA-1024 was breakable.

DNSSEC's main excuse for sticking to RSA-1024: speed. "Tradeoff between the risk of key compromise and performance."

How to convince terrorists that secure crypto is too slow?

Many techniques: obsolete data, incompetent benchmarks, fraud.

Example:

"PRESERVE contributes to the security and privacy of future vehicle-to-vehicle and vehicle-to-infrastructure communication systems by addressing critical issues like performance, scalability, and deployability of V2X security systems."

preserve-project.eu

"[In] most driving situations the packet rates do not exceed 750 packets per second. Only the maximum highway scenario goes well beyond this value (2,265 packets per second).

Processing 1,000 packets per second at 1 ms per packet can hardly be met by current hardware. A Pentium D 3.4 GHz processor needs about 3.3 seconds to verify a encrypted packet, i.e., a dedicated cryptographic co-processor is likely to be necessary."
How to convince terrorists that secure crypto is too slow?

Many techniques: obsolete data, incompetent benchmarks, fraud.

Example:

“PRESERVE contributes to the security and privacy of future vehicle-to-vehicle and vehicle-to-infrastructure communication systems by addressing critical issues like performance, scalability, and deployability of V2X security systems.”

preserve-project.eu

“In most driving situations: the packet rates do not exceed 750 packets per second. Only the maximum highway scenario goes well beyond this value (2,265 packets per second).

Processing 1,000 packets per second and processing each in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification … a dedicated cryptographic co-processor is likely to be necessary.”
2014: DNSSEC uses RSA-1024 to “secure” IP addresses. e.g. dnssec-deployment.org address is signed by RSA-1024. Fact: Analyses in 2003 concluded that RSA-1024 was breakable; e.g., 2003 Shamir–Tromer estimated 1 year, 10^7 USD. DNSSEC’s main excuse for sticking to RSA-1024: speed. “Tradeoff between the risk of key compromise and performance.”

How to convince terrorists that secure crypto is too slow?
Many techniques: obsolete data, incompetent benchmarks, fraud.
Example:
“PRESERVE contributes to the security and privacy of future vehicle-to-vehicle and vehicle-to-infrastructure communication systems by addressing critical issues like performance, scalability, and deployability of V2X security systems.”
preserve-project.eu

“[In] most driving situations, the packet rates do not exceed 750 packets per second. Only the maximum highway scenario goes well beyond this value (2,265 packets per second).

Processing 1,000 packets per second and processing each packet in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification . . . a dedicated cryptographic co-processor is likely to be necessary.”
How to convince terrorists that secure crypto is too slow?

Many techniques: obsolete data, incompetent benchmarks, fraud.

Example:
“PRESERVE contributes to the security and privacy of future vehicle-to-vehicle and vehicle-to-infrastructure communication systems by addressing critical issues like performance, scalability, and deployability of V2X security systems.”

preserve-project.eu

“[In] most driving situations . . . the packet rates do not exceed 750 packets per second. Only the maximum highway scenario . . . goes well beyond this value (2,265 packets per second). . . .

Processing 1,000 packets per second and processing each in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification . . . a dedicated cryptographic co-processor is likely to be necessary.”
How to convince terrorists that secure crypto is too slow?

Many techniques: obsolete data, incompetent benchmarks, fraud.

Example:

"PRESERVE contributes to the security and privacy of future vehicle-to-vehicle and vehicle-to-infrastructure communication systems by addressing critical issues like performance, scalability, and deployability of V2X security systems."

[preserve-project.eu](preserve-project.eu)

"[In] most driving situations the packet rates do not exceed 750 packets per second. Only the maximum highway scenario goes well beyond this value (2,265 packets per second). . . .

Processing 1,000 packets per second and processing each in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification . . . a dedicated cryptographic co-processor is likely to be necessary."

Compare to "NEON crypto" on 1GHz Cortex-A8 core:

- 5.48 cycles/byte (1.4 Gbps),
- 2.30 cycles/byte (3.4 Gbps)
for Salsa20, Poly1305.

- 498349 cycles (2000/second),
- 624846 cycles (1600/second)
for Curve25519 DH, verify."
How to convince terrorists that secure crypto is too slow?

Many techniques: obsolete data, incompetent benchmarks, fraud.

“[In] most driving situations . . . the packet rates do not exceed 750 packets per second. Only the maximum highway scenario . . . goes well beyond this value (2,265 packets per second). . . . Processing 1,000 packets per second and processing each in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification . . . a dedicated cryptographic co-processor is likely to be necessary.”

Compare to “NEON crypto” on 1GHz Cortex-A8:
5.48 cycles/byte (1.4 Gbps),
2.30 cycles/byte (3.4 Gbps)
for Salsa20, Poly1305.
498349 cycles (2000/second),
624846 cycles (1600/second)
for Curve25519 DH, verify.
“[In] most driving situations … the packet rates do not exceed 750 packets per second. Only the maximum highway scenario … goes well beyond this value (2,265 packets per second). … Processing 1,000 packets per second and processing each in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification … a dedicated cryptographic co-processor is likely to be necessary.”

Compare to “NEON crypto” on 1GHz Cortex-A8 core: 5.48 cycles/byte (1.4 Gbps), 2.30 cycles/byte (3.4 Gbps) for Salsa20, Poly1305. 498349 cycles (2000/second), 624846 cycles (1600/second) for Curve25519 DH, verify.
“[In] most driving situations . . . the packet rates do not exceed 750 packets per second. Only the maximum highway scenario . . . goes well beyond this value (2,265 packets per second). . . .

Processing 1,000 packets per second and processing each in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification . . . a dedicated cryptographic co-processor is likely to be necessary.”

Compare to “NEON crypto” on 1GHz Cortex-A8 core:
5.48 cycles/byte (1.4 Gbps),
2.30 cycles/byte (3.4 Gbps)
for Salsa20, Poly1305.
498349 cycles (2000/second),
624846 cycles (1600/second)
for Curve25519 DH, verify.
“[In] most driving situations . . . the packet rates do not exceed 750 packets per second. Only the maximum highway scenario . . . goes well beyond this value (2,265 packets per second). . . .

Processing 1,000 packets per second and processing each in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification . . . a dedicated cryptographic co-processor is likely to be necessary.”

Compare to “NEON crypto” on 1GHz Cortex-A8 core:
5.48 cycles/byte (1.4 Gbps),
2.30 cycles/byte (3.4 Gbps)
for Salsa20, Poly1305.
498349 cycles (2000/second),
624846 cycles (1600/second)
for Curve25519 DH, verify.

1GHz Cortex-A8 was high-end smartphone core in 2010: e.g.,
Samsung Exynos 3110 (Galaxy S);
TI OMAP3630 (Motorola Droid X); Apple A4 (iPad 1/iPhone 4).
“[In] most driving situations . . . the packet rates do not exceed 750 packets per second. Only the maximum highway scenario . . . goes well beyond this value (2,265 packets per second). . . .

Processing 1,000 packets per second and processing each in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification . . . a dedicated cryptographic co-processor is likely to be necessary.”

Compare to “NEON crypto” on 1GHz Cortex-A8 core:
5.48 cycles/byte (1.4 Gbps),
2.30 cycles/byte (3.4 Gbps)
for Salsa20, Poly1305.
498349 cycles (2000/second),
624846 cycles (1600/second)
for Curve25519 DH, verify.

1GHz Cortex-A8 was high-end smartphone core in 2010: e.g., Samsung Exynos 3110 (Galaxy S); TI OMAP3630 (Motorola Droid X); Apple A4 (iPad 1/iPhone 4).

2013: Allwinner A13, $5 in bulk.
In most driving situations, the packet rates do not exceed 750 packets per second. Only the maximum highway scenario goes well beyond this value (2,265 packets per second). Processing 1,000 packets per second and processing each in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification. A dedicated cryptographic co-processor is likely to be necessary.

Compare to “NEON crypto” on 1GHz Cortex-A8 core:
- 5.48 cycles/byte (1.4 Gbps),
- 2.30 cycles/byte (3.4 Gbps)
for Salsa20, Poly1305.
- 498349 cycles (2000/second),
- 624846 cycles (1600/second)
for Curve25519 DH, verify.

1GHz Cortex-A8 was high-end smartphone core in 2010: e.g., Samsung Exynos 3110 (Galaxy S); TI OMAP3630 (Motorola Droid X); Apple A4 (iPad 1/iPhone 4).

2013: Allwinner A13, $5 in bulk.

What if the terrorists hear about fast secure crypto? Yikes!
Similar to constant-time story. Don’t standardize good crypto. Discourage use of good crypto.
Most driving situations do not exceed 750 packets per second. Only the maximum highway scenario goes well beyond this value (2,265 packets per second).

Processing 1,000 packets per second and processing each in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification: a dedicated cryptographic co-processor is likely to be necessary.

Compare to "NEON crypto" on 1GHz Cortex-A8 core:
- 5.48 cycles/byte (1.4 Gbps),
- 2.30 cycles/byte (3.4 Gbps)
for Salsa20, Poly1305.
- 498349 cycles (2000/second),
- 624846 cycles (1600/second)
for Curve25519 DH, verify.

1GHz Cortex-A8 was high-end smartphone core in 2010: e.g., Samsung Exynos 3110 (Galaxy S); TI OMAP3630 (Motorola Droid X); Apple A4 (iPad 1/iPhone 4).

2013: Allwinner A13, $5 in bulk.

What if the terrorists hear about fast secure crypto? Yikes!

Similar to constant-time story. Don’t standardize good crypto. Discourage use of.
In most driving situations, the packet rates do not exceed 750 packets per second. Only the maximum highway scenario goes well beyond this value (2,265 packets per second). Processing 1,000 packets per second and processing each in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification: a dedicated cryptographic co-processor is likely to be necessary.

Compare to “NEON crypto” on 1GHz Cortex-A8 core:
- 5.48 cycles/byte (1.4 Gbps),
- 2.30 cycles/byte (3.4 Gbps) for Salsa20, Poly1305.
- 498349 cycles (2000/second),
- 624846 cycles (1600/second) for Curve25519 DH, verify.

1GHz Cortex-A8 was high-end smartphone core in 2010: e.g.,
- Samsung Exynos 3110 (Galaxy S);
- TI OMAP3630 (Motorola Droid X); Apple A4 (iPad 1/iPhone 4).

2013: Allwinner A13, $5 in bulk.

What if the terrorists hear about fast secure crypto? Yikes!

Similar to constant-time story.

Don’t standardize good crypto.

Discourage use of good crypto.
Compare to “NEON crypto” on 1GHz Cortex-A8 core:
5.48 cycles/byte (1.4 Gbps),
2.30 cycles/byte (3.4 Gbps)
for Salsa20, Poly1305.
498349 cycles (2000/second),
624846 cycles (1600/second)
for Curve25519 DH, verify.

1GHz Cortex-A8 was high-end smartphone core in 2010: e.g.,
Samsung Exynos 3110 (Galaxy S);
TI OMAP3630 (Motorola Droid X);
Apple A4 (iPad 1/iPhone 4).

2013: Allwinner A13, $5 in bulk.

What if the terrorists hear about fast secure crypto?
Yikes!

Similar to constant-time story.
Don’t standardize good crypto.
Discourage use of good crypto.
Compare to “NEON crypto” on 1GHz Cortex-A8 core:
5.48 cycles/byte (1.4 Gbps),
2.30 cycles/byte (3.4 Gbps)
for Salsa20, Poly1305.
498349 cycles (2000/second),
624846 cycles (1600/second)
for Curve25519 DH, verify.

1GHz Cortex-A8 was high-end smartphone core in 2010: e.g.,
Samsung Exynos 3110 (Galaxy S);
TI OMAP3630 (Motorola Droid X); Apple A4 (iPad 1/iPhone 4).
2013: Allwinner A13, $5 in bulk.

What if the terrorists hear about fast secure crypto?
Yikes!

Similar to constant-time story.
Don’t standardize good crypto.
Discourage use of good crypto.
If the good crypto persists, try to bury it behind a huge menu of bad options.
Advertise “cryptographic agility”; actually cryptographic fragility.
Pretend that this “agility” justifies using breakable crypto.
What if the terrorists hear about fast secure crypto? Yikes!

Similar to constant-time story. Don’t standardize good crypto. Discourage use of good crypto.

If the good crypto persists, try to bury it behind a huge menu of bad options. Advertise “cryptographic agility”; actually cryptographic fragility. Pretend that this “agility” justifies using breakable crypto.

Precomputed signatures
Try to build cryptographic fragility into many layers of the system. e.g. Complicate the protocols.
ON crypto

1GHz Cortex-A8 core:

5.48 cycles/byte (1.4 Gbps),
2.30 cycles/byte (3.4 Gbps)

1GHz Cortex-A8 was high-end smartphone core in 2010: e.g.,
Samsung Exynos 3110 (Galaxy S);
TI OMAP3630 (Motorola Droid X);
Apple A4 (iPad 1/iPhone 4).

2013: Allwinner A13, $5 in bulk.

What if the terrorists hear about fast secure crypto?
Yikes!

Similar to constant-time story.
Don’t standardize good crypto.
Discourage use of good crypto.

If the good crypto persists,
try to bury it behind a huge menu of bad options.
Advertise “cryptographic agility”;
actually cryptographic fragility.
Pretend that this “agility” justifies using breakable crypto.

Precomputed signatures

Try to build cryptographic fragility into many layers of the system.
e.g. Complicate the protocols.
Compare to "NEON crypto" on 1GHz Cortex-A8 core:
5.48 cycles/byte (1.4 Gbps),
2.30 cycles/byte (3.4 Gbps)
for Salsa20, Poly1305.

498349 cycles (2000/second),
624846 cycles (1600/second)
for Curve25519 DH, verify.

1GHz Cortex-A8 was high-end smartphone core in 2010: e.g., Samsung Exynos 3110 (Galaxy S);
TI OMAP3630 (Motorola Droid X); Apple A4 (iPad 1/iPhone 4).

2013: Allwinner A13, $5 in bulk.

What if the terrorists hear about fast secure crypto?
Yikes!

Similar to constant-time story.
Don't standardize good crypto.
Discourage use of good crypto.

If the good crypto persists,
try to bury it behind a huge menu of bad options.
Advertise “cryptographic agility”; actually cryptographic fragility.
Pretend that this “agility” justifies using breakable crypto.

Precomputed signatures
Try to build cryptographic fragility into many layers of the system.
e.g. Complicate the protocols.

Precomputed signatures
Try to build cryptographic fragility into many layers of the system.
e.g. Complicate the protocols.
What if the terrorists hear about fast secure crypto?
Yikes!

Similar to constant-time story.
Don’t standardize good crypto.
Discourage use of good crypto.

If the good crypto persists,
try to bury it behind
a huge menu of bad options.
Advertise “cryptographic agility”;
actually cryptographic fragility.
Pretend that this “agility”
justifies using breakable crypto.

Precomputed signatures
Try to build cryptographic fragility
into many layers of the system.
e.g. Complicate the protocols.
What if the terrorists hear about fast secure crypto?
Yikes!

Similar to constant-time story.
Don’t standardize good crypto.
Discourage use of good crypto.

If the good crypto persists,
try to bury it behind
a huge menu of bad options.
Advertise “cryptographic agility”;
actually cryptographic fragility.
Pretend that this “agility”
justifies using breakable crypto.

Precomputed signatures
Try to build cryptographic fragility
into many layers of the system.

e.g. Complicate the protocols.

Split cryptographic security
into “the easy problem”
of protecting integrity
and “the hard problem”
of protecting confidentiality.
What if the terrorists hear about fast secure crypto? Yikes!

Similar to constant-time story. Don’t standardize good crypto. Discourage use of good crypto.

If the good crypto persists, try to bury it behind a huge menu of bad options. Advertise “cryptographic agility”; actually cryptographic fragility. Pretend that this “agility” justifies using breakable crypto.

Precomputed signatures
Try to build cryptographic fragility into many layers of the system.

E.g. Complicate the protocols.

Split cryptographic security into “the easy problem” of protecting integrity and “the hard problem” of protecting confidentiality.

E.g. Argue against encrypted SNI since DNS is unencrypted, and argue against encrypted DNS since SNI is unencrypted.
What if the terrorists hear about fast secure crypto? Yikes!

Similar to constant-time story. Don't standardize good crypto. Discourage use of good crypto. If the good crypto persists, try to bury it behind a huge menu of bad options. Pretend that this “agility” justifies using breakable crypto.

Precomputed signatures
Try to build cryptographic fragility into many layers of the system.

e.g. Complicate the protocols.

Split cryptographic security into “the easy problem” of protecting integrity and “the hard problem” of protecting confidentiality.

e.g. argue against encrypted SNI since DNS is unencrypted, and argue against encrypted DNS since SNI is unencrypted.

Precomputed signatures
Try to build cryptographic fragility into many layers of the system.

e.g. Complicate the protocols.

Split cryptographic security into “the easy problem” of protecting integrity and “the hard problem” of protecting confidentiality.

e.g. argue against encrypted SNI since DNS is unencrypted, and argue against encrypted DNS since SNI is unencrypted.
Precomputed signatures

Try to build cryptographic fragility into many layers of the system.

e.g. Complicate the protocols.

Split cryptographic security into “the easy problem” of protecting integrity and “the hard problem” of protecting confidentiality.

e.g. argue against encrypted SNI since DNS is unencrypted, and argue against encrypted DNS since SNI is unencrypted.

Solve “the easy problem” by precomputing signatures.

Insist that the protocol allow precomputation “for speed”.

e.g. DNSSEC.
Precomputed signatures
Try to build cryptographic fragility into many layers of the system.
e.g. Complicate the protocols.

Split cryptographic security into “the easy problem” of protecting integrity and “the hard problem” of protecting confidentiality.
e.g. argue against encrypted SNI since DNS is unencrypted, and argue against encrypted DNS since SNI is unencrypted.

Solve “the easy problem” by precomputing signatures. Insist that the protocol allow precomputation “for speed”. e.g. DNSSEC.
Precomputed signatures

Try to build cryptographic fragility into many layers of the system.

e.g. Complicate the protocols.

Split cryptographic security into “the easy problem” of protecting integrity and “the hard problem” of protecting confidentiality.

e.g. argue against encrypted SNI since DNS is unencrypted, and argue against encrypted DNS since SNI is unencrypted.

Solve “the easy problem” by precomputing signatures. Insist that the protocol allow precomputation “for speed”.

e.g. DNSSEC.
Precomputed signatures

Try to build cryptographic fragility into many layers of the system.

e.g. Complicate the protocols.

Split cryptographic security into “the easy problem” of protecting integrity and “the hard problem” of protecting confidentiality.

e.g. argue against encrypted SNI since DNS is unencrypted, and argue against encrypted DNS since SNI is unencrypted.

Solve “the easy problem” by precomputing signatures.

Insist that the protocol allow precomputation “for speed”.

e.g. DNSSEC.

The protocol has trouble handling dynamically generated answers, and unpredictable questions; also, trouble guaranteeing freshness. Deployment hits many snags.
Precomputed signatures
Try to build cryptographic fragility into many layers of the system.
e.g. Complicate the protocols.

Split cryptographic security into “the easy problem” of protecting integrity and “the hard problem” of protecting confidentiality.
e.g. argue against encrypted SNI since DNS is unencrypted, and argue against encrypted DNS since SNI is unencrypted.

Solve “the easy problem” by precomputing signatures. Insist that the protocol allow precomputation “for speed”. e.g. DNSSEC.

The protocol has trouble handling dynamically generated answers, and unpredictable questions; also, trouble guaranteeing freshness. Deployment hits many snags.

Argue that it’s too early to look at “the hard problem” when most data is still unsigned.
Precomputed signatures

Try to build cryptographic fragility into many layers of the system.

- Complicate the protocols.

Cryptographic security

- Solve “the easy problem” by precomputing signatures.
- Insist that the protocol allow precomputation “for speed”.
  e.g. DNSSEC.

The protocol has trouble handling dynamically generated answers, and unpredictable questions; also, trouble guaranteeing freshness. Deployment hits many snags.

Argue that it’s too early to look at “the hard problem” when most data is still unsigned.

More strategies

- Divert “crypto” funding and human resources into activities that don’t threaten mass surveillance.
- Set up centralized systems encrypting data to companies that collaborate with us.
- More distraction: build systems breakable by active attacks.
- Declare crypto success without encrypting the Internet.
Precomputed signatures
Try to build cryptographic fragility into many layers of the system.

e.g. Complicate the protocols.

Split cryptographic security into "the easy problem" of protecting integrity and "the hard problem" of protecting confidentiality.

e.g. argue against encrypted SNI since DNS is unencrypted, and argue against encrypted DNS since SNI is unencrypted.

Solve "the easy problem" by precomputing signatures. Insist that the protocol allow precomputation "for speed".
e.g. DNSSEC.

The protocol has trouble handling dynamically generated answers, and unpredictable questions; also, trouble guaranteeing freshness.

Deployment hits many snags.

Argue that it's too early to look at "the hard problem" when most data is still unsigned.

More strategies
Divert "crypto" funding and human resources into activities that don't threaten mass surveillance.

Set up centralized systems encrypting data to companies that collaborate with us.

More distraction: build systems breakable by active attacks.

Declare crypto success without encrypting the Internet.
Precomputed signatures
Try to build cryptographic fragility into many layers of the system.
e.g. Complicate the protocols.
Split cryptographic security into “the easy problem” of protecting integrity and “the hard problem” of protecting confidentiality.
e.g. argue against encrypted SNI since DNS is unencrypted, and argue against encrypted DNS since SNI is unencrypted.

Solve “the easy problem” by precomputing signatures.
Insist that the protocol allow precomputation “for speed”. e.g. DNSSEC.
The protocol has trouble handling dynamically generated answers, and unpredictable questions; also, trouble guaranteeing freshness.
Deployment hits many snags.
Argue that it’s too early to look at “the hard problem” when most data is still unsigned.

More strategies
Divert “crypto” funding and human resources into activities that don’t threaten mass surveillance.
Set up centralized systems encrypting data to companies that collaborate with us.
More distraction: build systems breakable by active attacks.
Declare crypto success without encrypting the Internet.

SNI
DNS
Solve “the easy problem” by precomputing signatures. Insist that the protocol allow precomputation “for speed”. e.g. DNSSEC.

The protocol has trouble handling dynamically generated answers, and unpredictable questions; also, trouble guaranteeing freshness. Deployment hits many snags.

Argue that it’s too early to look at “the hard problem” when most data is still unsigned.

More strategies
Divert “crypto” funding and human resources into activities that don’t threaten mass surveillance.

Set up centralized systems encrypting data to companies that collaborate with us.

More distraction: build systems breakable by active attacks.

Declare crypto success without encrypting the Internet.