badkeys

Unsichere kryptographische Schlüssel

https://badkeys.info/

Hanno Böck

Public-Key-Kryptographie

🔑 Öffentlicher Schlüssel

🔑 Privater Schlüssel

Verschlüsselung (veraltet)

Verschlüsseln mit öffentlichem Schlüssel🔑

Entschlüsseln mit privatem Schlüssel🔑

Digitale Signaturen

Signieren mit privatem Schlüssel🔑

Prüfen mit öffentlichem Schlüssel🔑

Public-Key-Verfahren

- RSA
- ECDSA, Ed25519/Ed448 (elliptische Kurven)
- ML-DSA, ML-KEM (Post-Quanten-Krypto)

Wo wird Public-Key-Kryptographie genutzt?

TLS, HTTPS, SSH, DKIM, DNSSEC, ...

Fahrkarten, E-Perso, Impfzertifikate, ...

Public-Key-Kryptographie ist nur sicher, wenn der private Schlüssel geheim ist

Sicherheitslücke in keypair / GitKraken (2021)

"There is no haveibeenpwned for public keys as far as I know"

("Es gibt soweit ich weiß kein haveibeenpwned für öffentliche Schlüssel")

user jornane on lobste.rs, 10/2021

haveibeenpwned für Public Keys?

Klingt nach einer guten Idee!

badkeys

Checking cryptographic public keys for known vulnerabilities

Enter Key:

```
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEA3+7n3nVVncZkM25RdNknGuAfkKMwcYaWqQGN4C01zGQlpJvDmXDd
VM+YNi4pZwRACs21V+h6LLy7mieYCWZy9NV/AfjovxZEInrCX3nyHgBxtYaV+bta
jFzFE4T3PJvuE5hc6Kx/KawU1NvE+kvQAirGDcVQe500rEcePgdezBh9wSA0yk/q
Pbc8hgrR6z255RGodzLgJi6v7uaXdvQlhDuIwjITIcdakspIX09xdCI1IzLkEdHX
dr4gutR8/D0UoR3RSfiWLwc4AcdAGvJ0X0HK3tCawPFxGoU3YkLIOIa0XYHTlIqu
Dyq3WaxIJjM/tDRYlBbYl1UF+qHcy4BDjQIDAQAB
-----END RSA PUBLIC KEY-----
```

OK CLEAR

https://badkeys.info/

CVE-2008-0166

Debian-OpenSSL-Bug

Durch einen fehlerhaften Patch funktionierte der Zufallszahlengenerator nicht korrekt und verwendete lediglich die Prozess-ID (PID) als Zufallsquelle

Testen, ob Schlüssel für Debian-OpenSSL-Bug verwundbar ist

(vor badkeys)

- Alte Skripte, die auf modernen Systemen nicht funktionieren
- Unvollständige Sammlungen oder Hash-Listen von betroffenen Schlüsseln
- Verwirrende, unvollständige oder falsche Informationen

Schlüsselvarianten Debian-OpenSSL-Bug

- PID (0 bis 32768)
- OpenSSL oder OpenSSH
- CPU-Architektur
- Schlüsselgröße
- Ältere oder neuere Version des betroffenen OpenSSL-Pakets
- Konfigurationsunterschiede

Let's Encrypt

Log In

Issuing for common RSA key sizes only

API Announcements

jsha Let's Encrypt engineer 1 Sep 2020

Effective 2020-09-17, we're requiring that all RSA keys for end-entity (leaf) certificates have a modulus of length 2048, 3072, or 4096.

Let's Encrypt (2020)

ECDSA

An mehreren Stellen wird erwähnt, dass die betroffene OpenSSL-Version kein ECDSA unterstützt.

Das stimmt nicht!

Debian Weak Keys and ECDSA, Hanno Böck (2022)

Alle Varianten des Debian-OpenSSL-Bugs zu erkennen ist unpraktikabel, badkeys erkennt aber alle plausiblen Varianten

https://github.com/badkeys/debianopenssl

Warum redet der so viel über eine 17 Jahre alte Sicherheitslücke?

Matt Palmer Mar 7, 2020, 3:48:48 AM

to mozilla-dev-s...@lists.mozilla.org

(Pre) Certificate https://crt.sh/?id=2531502044 has been issued with a known weak key, specifically Debian weak key 2048/i386/rnd/pid17691. I believe this issuance to be in contravention of SSL.com's CPS, version 1.8, section 6.1.1.2, which states "SSL.com shall reject a certificate request if the request has a known weak Private Key".

- Matt

Matt Palmer (2020)

DKIM

DomainKeys Identified Mail

DKIM E-Mail header

DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;
d=hboeck.de; s=key1; t=1715197611;
bh=Z9fPSuWvmaUL/fgn9g0k2ORYPJe3Y3Vc5NiKvQJXc2w=;
h=Date:From:To:Subject:Message-ID:MIME-Version:Content-Type:
Content-Transfer-Encoding; b=TNyZHQd[...]

TXT record *key1._domainkey.hboeck.de*

v=DKIM1; k=rsa; p=MIIBIjANBgkqhkiG9w0BAQE[...]

Scan von DKIM-Keys

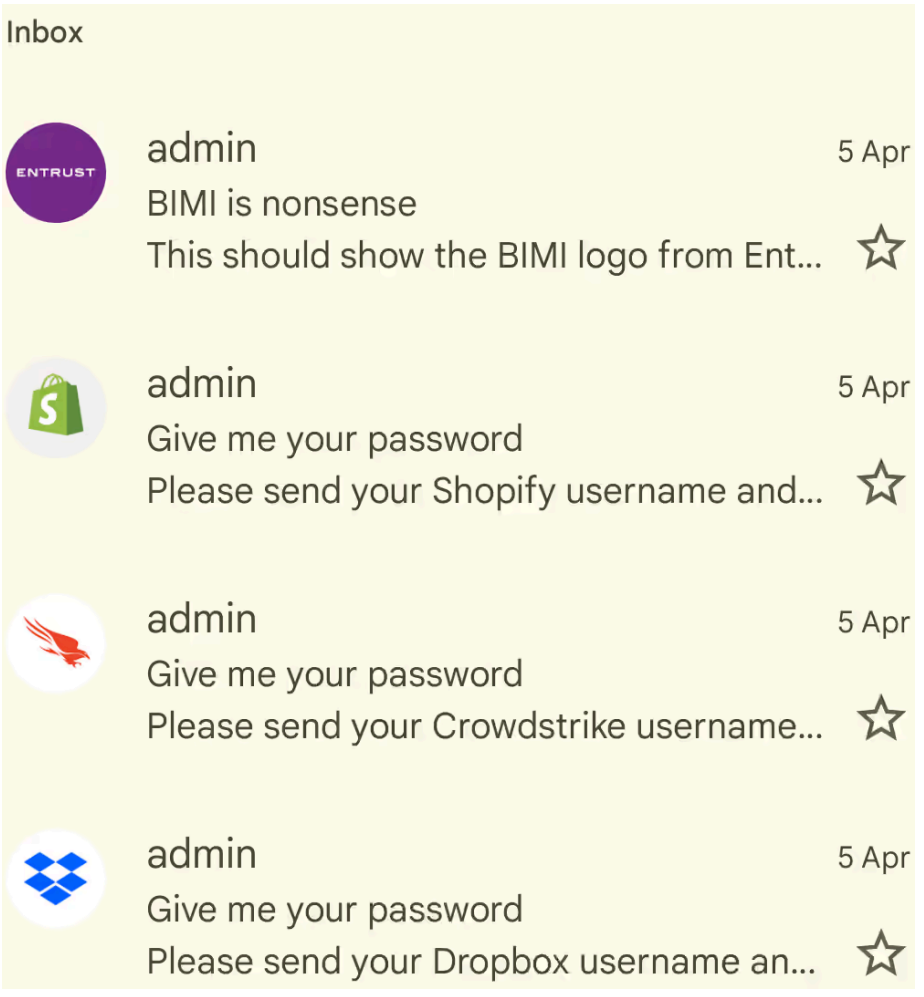
Benötigt passende DKIM-Selektoren, extrahieren aus bestehenden Mails oder gängige Selektoren (key, dkim, ...) raten / bruteforce

DKIM Scan mit badkeys (2024)

0,24 % der gescannten DKIM-Records betroffen von Debian-OpenSSL-Bug (2024!)

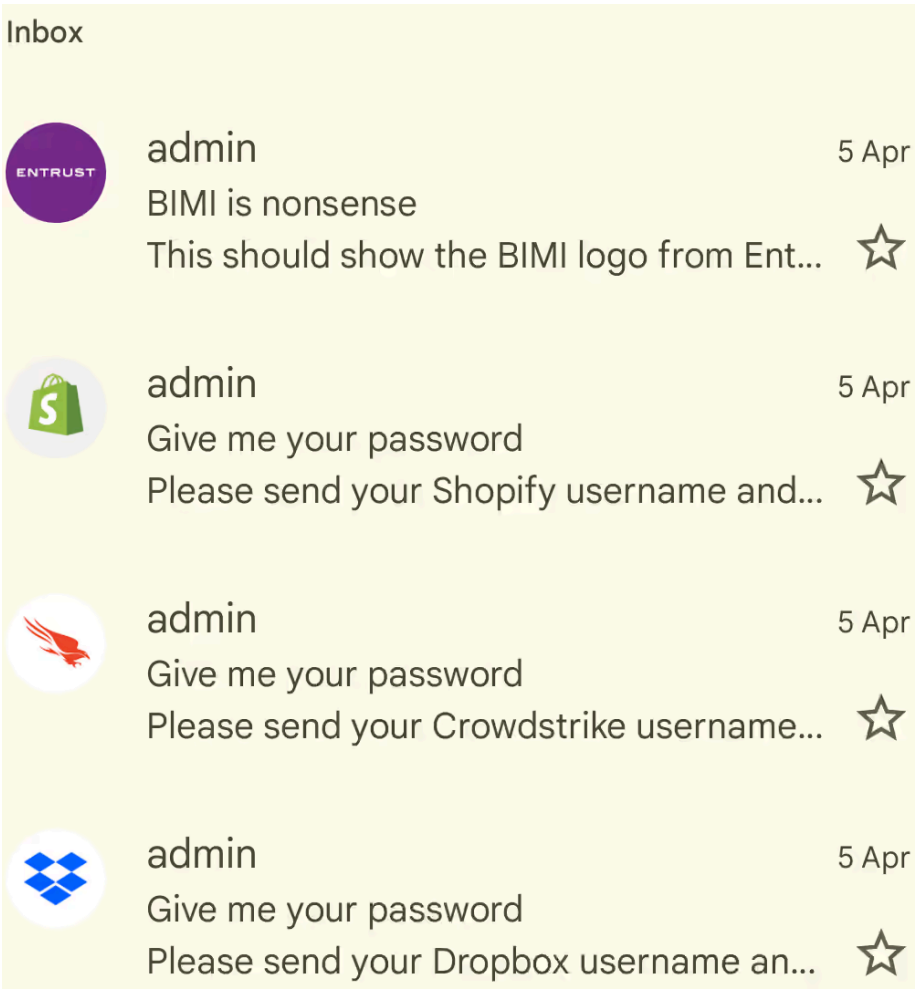
Betroffene Hosts

@cisco.com, @oracle.com, @skype.net, @github.partners, @partner.crowdstrike.com, @partners.dropbox.com, @1password.com


Inbox
admin
5 Apr
BIMI is nonsense
This should show the BIMI logo from Ent...
admin
5 Apr
Give me your password
Please send your Shopify username and...
admin
5 Apr
Give me your password
Please send your Crowdstrike username...
admin
5 Apr
Give me your password
Please send your Dropbox username an...

BIMI

Brand Indicators for Message Identification


Inbox
admin
5 Apr
BIMI is nonsense
This should show the BIMI logo from Ent...
admin
5 Apr
Give me your password
Please send your Shopify username and...
admin
5 Apr
Give me your password
Please send your Crowdstrike username...
admin
5 Apr
Give me your password
Please send your Dropbox username an...

Warum waren relativ viele DKIM-Setups von einem 16 Jahre alten Bug betroffen?

2006: Debian-OpenSSL-Paket mit Bug veröffentlicht

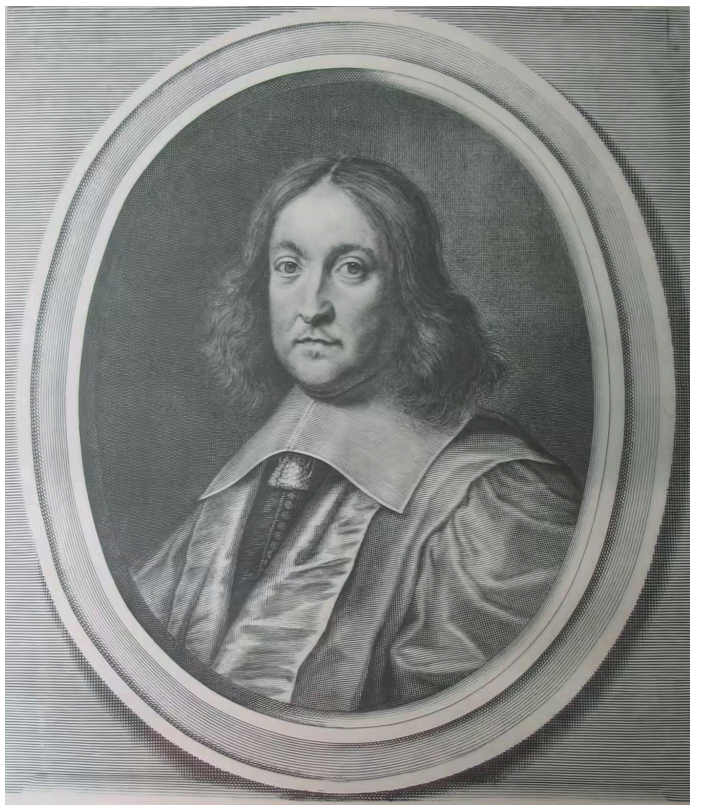
2007: RFC 4871 (DKIM)

2008: Debian-OpenSSL-Bug wird entdeckt und gefixt

Debian OpenSSL Bug und DKIM

https://16years.secvuln.info/

Mehr Infos und Vortragsvideo (MiniDebConf)



Fermat-Angriff (1643)

RSA-Schlüssel

Öffentlicher Schlüssel: N, e

Privater Schlüssel: N, e, d, p, q, dP, dQ, invQ

Faktorisierung

$$N = p \cdot q$$

N: Modulus (öffentlich),
p/q: Primzahlen (privat)

$$N = p \cdot q$$

Aus p oder q und dem öffentlichen Schlüssel kann man den gesamten private Schlüssel berechnen

Die Sicherheit von RSA hängt davon ab, dass Faktorisierung "schwierig" ist

Fermat-Faktorisierung

$$N = p \cdot q$$

a: Mitte zwischen p und q

b: Abstand zwischen a und p/q

$$N = (a + b) \cdot (a - b)$$

$$N = (a + b) \cdot (a - b)$$

$$N = a^2 - b^2$$

$$b^2 = a^2 - N$$

$$b^2 = a^2 - N$$

Wir raten Werte von a (Start: √N) und prüfen, ob das Resultat eine Quadratzahl ist

```
a = gmpy2.isqrt(n)
while not gmpy2.is_square(a**2 - n):
    a += 1
bsq = a**2 - n
b = gmpy2.isqrt(bsq)
p = a + b
q = a - b
```

Fermat-Faktorisierung ist schnell, wenn Primzahlen nah beieinander liegen

1993 - 1996

Rechtsextreme Bombenanschläge in Österreich (Bajuwarischen Befreiungsarmee, Franz Fuchs)

DECHIFFRIERANWEISUNG:

Die Seiten 1 bis 15 enthalten Serien von Chiffratzahlen mit jeweils 241 bis 243 Dezimalstellen.

Die Klartextzahlen wurden chiffriert mit der Formel

Chiffratzahl = Rest von [(Klartextzahl hoch Exponent) geteilt durch (Primzahl1 mal Primzahl2)]

mit

Exponent =
50807531083515900981263396917441112349672885967273707669513982618625764758133748152167669282510298280822207623874775350440 7

und

(Primzahl1 mal Primzahl2) =
63054821507012954715671833249588963223443414541197127588837698760326022525278792613527673894410568910003629553586814142438653640364957870769912818949143213863190059077472921499001536910276096488477634484971781148430952891504011795209806188688 1.

Durch wiederholte Chiffrierung nach dieser Formel oder durch Umkehrung dieser Formel nach den bekannten Verfahren kommt man wieder zur Klartextzahl. Zur Lösung dieser nach simpler Hauptschulmathematik aussehenden Formel, die nicht einmal technisch anwendbar ist, wurden in den letzten Jahren Supercomputer mit spezieller Schaltkreisarchitektur gebaut.

Verschlüsselte Nachricht damals von Hans Dobbertin geknackt

Klaus Schmeh (2022)

Scan von TLS-Zertifikaten mit Fermat-Faktorisierung (2022)

CVE-2022-26320

Drucker von Canon und Fujifilm (TLS-Bibliothek von Rambus) erstellten verwundbare Zertifikate

Gemeinsame Primzahlen

$$N = p \cdot q$$

Angenommen, wir haben zwei RSA-Keys mit einer gemeinsamen und einer unterschiedlichen Primzahl

$$N_1 = p_1 \cdot q_1$$

$$N_2 = p_2 \cdot q_1$$

Größter gemeinsamer Teiler

$$N_1 = p_1 \cdot q_1 \qquad N_2 = p_2 \cdot q_1$$

$$ggT(N_1, N_2) = q_1$$

BatchGCD

Größte gemeinsame Teiler von vielen Zahlen

2012 finden zwei Teams zahlreiche RSA-Schlüssel mit gemeinsamen Primfaktoren (Open-Source Code von Nadia Heninger unter factorable.net)

Heninger et al, Usenix (2012)

Lenstra et al (2012)

Warum passiert sowas?

Die betroffenen Schlüssel wurden überwiegend auf Embedded-Geräten unter Linux erstellt

Zufallszahlengenerator des Betriebssystems

Tastatureingaben, Festplatten-Zugriffszeiten, etc., werden genutzt, um den Zufallszahlengenerator zu initialisieren

1. Betriebssystem startet, Zufallszahlengenerator ist noch nicht sicher initialisiert
2. Startskript erzeugt RSA-Schlüssel beim Booten
3. Erste Primzahl wird erzeugt (unsicher, kann sich wiederholen)
4. Betriebssystem sammelt weitere Zufallsdaten
5. Zweite Primzahl wird erzeugt (sicher)

Nach dem Paper gab es zahlreiche Änderungen am Linux-Kernel, um derartige Szenarien zu vermeiden

Taiwain Citizen Digital Certificate

Bernstein et al (2013)

Sichere, zufällige Primzahl?

```
c000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000002f9
```

```
c9242492249292499249492449242492
2492924992494924492424922492924 9
9249492449242492249292499249492 4
4924249224929249924949244924 24e5
```

Certification Report

Federal Office for Information Security

BSI-DSZ-CC-0212-2004

for

Renesas AE45C1 (HD65145C1)
Smartcard Integrated Circuit
Version 01

from

Renesas Technology Corp.

BSI-zertifizierte Chipkarten

ROCA

Return of Coopersmith's Attack (2017)

ROCA

Sicherheitslücke in RSA-Chips von Infineon, Schlüssel lassen sich erkennen und (mit Aufwand) brechen

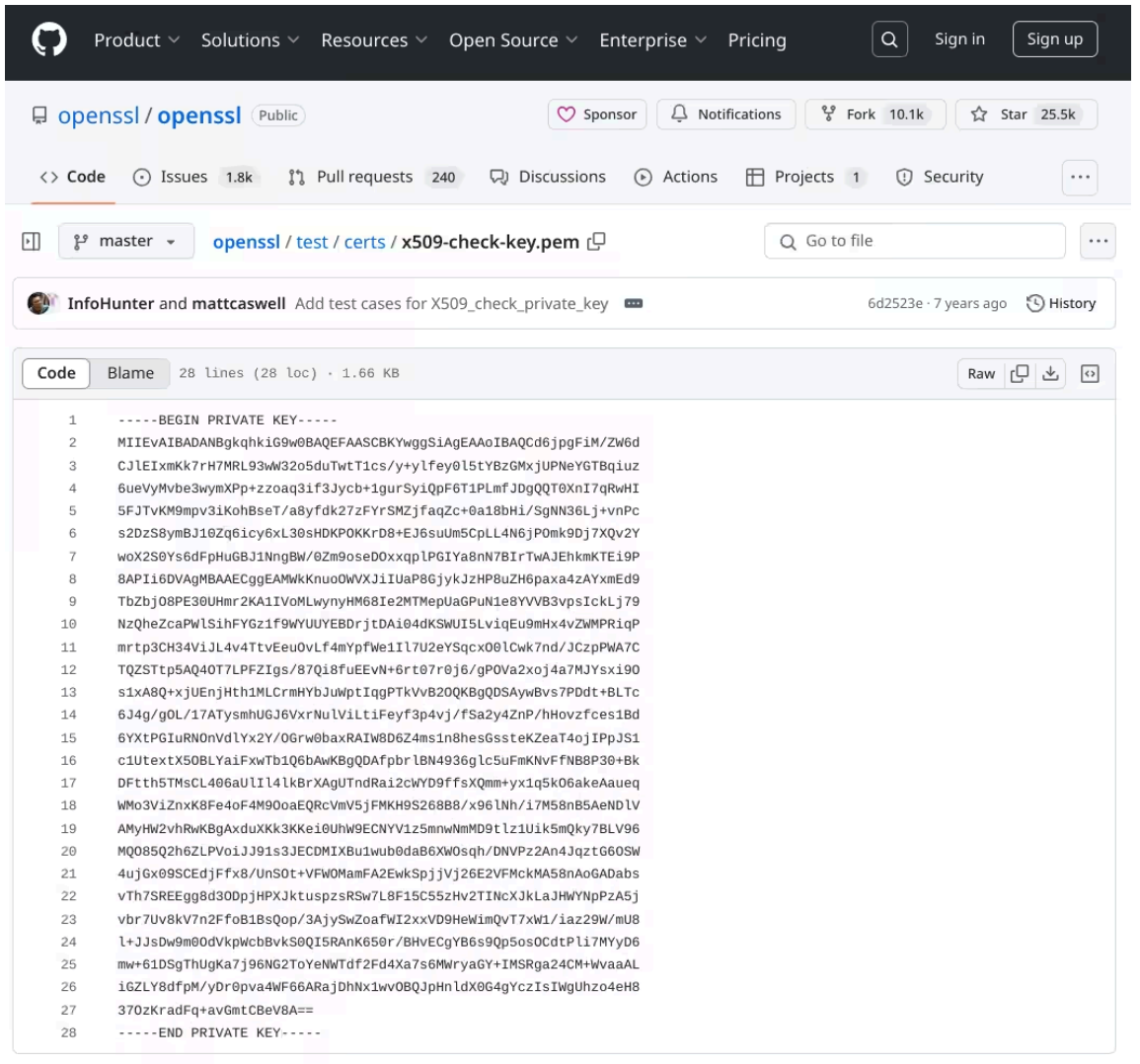
Wegen ROCA mussten in Estland Ausweise mit verwundbaren Chips ausgetauscht werden

Auch diese Chipkarten waren BSI-zertifiziert

badkeys-Fund Zertifikate von Yahoo mit ROCA-Keys (Ausgestellt 2021)

Public Private Keys

Öffentliche private Schlüssel


Product
Solutions
Resources
Open Source
Enterprise
Pricing
Sign in
Sign up
openssl / openssl
Public
Sponsor
Notifications
Fork 10.1k
Star 25.5k
Code
Issues 1.8k
Pull requests 240
Discussions
Actions
Projects 1
Security
master
openssl / test / certs / x509-check-key.pem
Go to file
InfoHunter and mattcaswell Add test cases for X509_check_private_key
6d2523e · 7 years ago
History
Code
Blame
28 lines (28 loc) · 1.66 KB
Raw
-----BEGIN PRIVATE KEY-----
MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQCd6jpgFiM/ZW6d
CJlEIxmKk7rH7MRL93wW32o5duTwtT1cs/y+ylfey0l5tYBzGMxjUPNeYGTBqiuz
6ueVyMvbe3wymXPp+zzoaq3if3Jycb+1gurSyiQpF6T1PLmfJDgQQT0XnI7qRwHI
5FJTvKM9mpv3iKohBseT/a8yfdk27zFYrSMZjfaqZc+0a18bHi/SgNN36Lj+vnPc
s2DzS8ymBJ10Zq6icy6xL30sHDKPOKKrD8+EJ6suUm5CpLL4N6jPOmk9Dj7XQv2Y
woX2S0Ys6dFpHuGBJ1NngBW/0Zm9oseDOxxqplPGIYa8nN7BIrTwAJEhkmKTEi9P
8APIi6DVAgMBAAECggEAMWkKnuoOWVXJiIUaP8GjykJzHP8uZH6paxa4zAYxmEd9
TbZbjO8PE30UHmr2KA1IVoMLwynyHM68Ie2MTMepUaGPuN1e8YVVB3vpsIckLj79
NzQheZcaPWlSihFYGz1f9WYUUYEBDrjtDAi04dKSWUI5LviqEu9mHx4vZWMPRiqP
mrtp3CH34ViJL4v4TtvEeuOvLf4mYpfWe1Il7U2eYSqcxO0lCwk7nd/JCzpPWA7C
TQZSTtp5AQ4OT7LPFZIgs/87Qi8fuEEvN+6rt07r0j6/gPOVa2xoj4a7MJYsxi9O
s1xA8Q+xjUEnjHth1MLCrmHYbJuWptIqgPTkVvB2OQKBgQDSAywBvs7PDdt+BLTc
6J4g/gOL/17ATysmhUGJ6VxrNulViLtiFeyf3p4vj/fSa2y4ZnP/hHovzfces1Bd
6YXtPGIuRNOnVdlYx2Y/OGrw0baxRAIW8D6Z4ms1n8hesGssteKZeaT4ojIPpJS1
c1UtextX5OBLYaiFxwTb1Q6bAwKBgQDAfpbrlBN4936glc5uFmKNvFfNB8P30+Bk
DFtth5TMsCL406aUlIl4lkBrXAgUTndRai2cWYD9ffsXQmm+yx1q5kO6akeAaueq
WMo3ViZnxK8Fe4oF4M9OoaEQRcVmV5jFMKH9S268B8/x96lNh/i7M58nB5AeNDlV
AMyHW2vhRwKBgAxduXKk3KKei0UhW9ECNYV1z5mnwNmMD9tlz1Uik5mQky7BLV96
MQO85Q2h6ZLPVoiJJ91s3JECDMIXBu1wub0daB6XWOsqh/DNVPz2An4JqztG6OSW
4ujGx09SCEdjFfx8/UnSOt+VFWOMamFA2EwkSpjjVj26E2VFMckMA58nAoGADabs
vTh7SREEgg8d3ODpjHPXJktuspzsRSw7L8F15C55zHv2TINcXJkLaJHWYNpPzA5j
vbr7Uv8kV7n2FfoB1BsQop/3AjySwZoafWI2xxVD9HeWimQvT7xW1/iaz29W/mU8
l+JJsDw9m0OdVkpWcbBvkS0QI5RAnK650r/BHvECgYB6s9Qp5osOCdtPli7MYyD6
mw+61DSgThUgKa7j96NG2ToYeNWTdf2Fd4Xa7s6MWryaGY+IMSRga24CM+WvaaAL
iGZLY8dfpM/yDr0pva4WF66ARajDhNx1wvOBQJpHnldX0G4gYczIsIWgUhzo4eH8
37OzKradFq+avGmtCBeV8A==
-----END PRIVATE KEY-----

Private Schlüssel, die man auf öffentlichen Webseiten finden kann, sind nicht sicher!

Bubcon Messenger

Bubcon-Security: Just Another Nightmare (2016, Nexus, CCC Hannover)

beA - besonderes elektronisches Anwaltspostfach (2017)

beA

Versuch 1: Von CA signiertes Zertifikat für bealocalhost.de mit Schlüssel in der Anwendung

beA

Versuch 2: Selbstsigniertes CA-Zertifikat mit Schlüssel in der Anwendung

Jenkins Docker-Images Hardcodierter SSH-Host-Key (CVE-2025-32754, CVE-2025-32755)

https://www.treasury.go.ke/treasury.go.ke.key

(Gefunden mit Tool snallygaster)

Es gibt viele Arten von öffentlichen privaten Schlüsseln

- Beispiele in Dokumentation
- Test-Keys in Softwarepaketen
- Geleakte Keys
- Vorgenerierte Keys in Firmware-Images, Containern, VM-Images
- Sicherheitslücken (Debian-OpenSSL-Bug, keypair-Bug)
- ...

Fortinet / FortiGate

CVE-2022-40684

Authentication Bypass

in FortiGate-Geräten

Bereits 2022 bekannt, dass diese aktiv ausgenutzt wurde

Januar 2025

~15.000 Konfigurationsdateien von FortiGate-Geräten werden auf BreachForums veröffentlicht

Diese Konfigurationsdateien enthalten private Schlüssel, allerdings sind diese mit einem Passwort verschlüsselt

Das Passwort ist auch in den Konfigurationsdateien, aber ebenfalls verschlüsselt

Das Schlüssel-Passwort wiederum war mit dem Passwort "Mary had a littl" verschlüsselt

FortiGate (2025)

Private Schlüssel für ~100 nach wie vor gültige TLS-Zertifikate und für ~300 ACME-Accounts (Let's Encrypt)

Vorfall war seit 2022 bekannt!

"Enterprise-Security"

OpenID Connect

https://example.com/.well-known/openid-configuration

JSON Web Key Öffentlich

```
{
"kty": "EC",
"crv": "P-256",
"x": "MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
"y": "4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM"
}
```

JSON Web Key Privat

```
{
"kty": "EC",
"crv": "P-256",
"x": "MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
"y": "4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
"d": "870MB6gfuTJ4HtUnUvYMyJpr5eUZNP4Bk43bVdj3eAE"
}
```

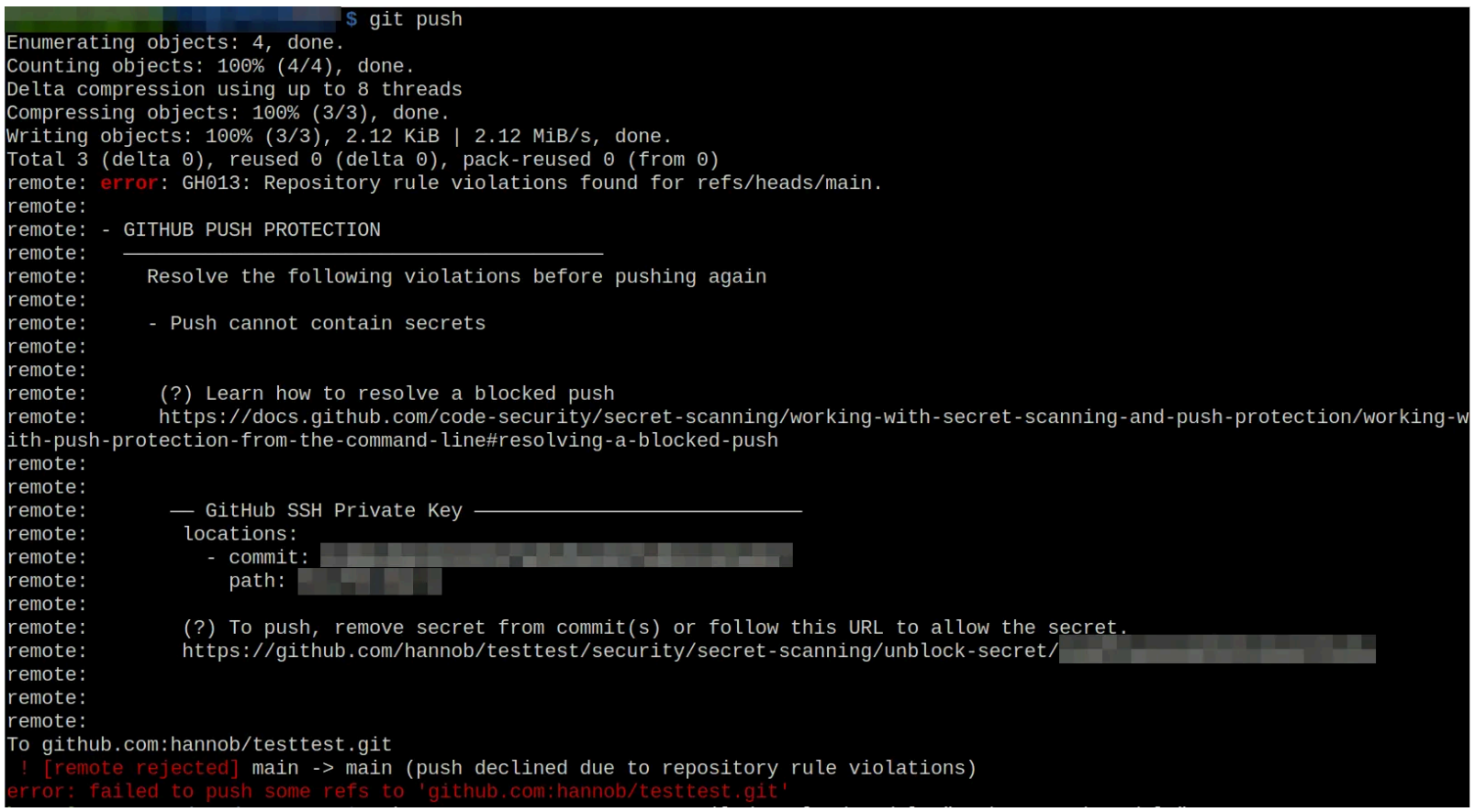
JSON Web Keys haben die Eigenschaft, dass das Format für private und öffentiche Schlüssel quasi identisch ist

Privater Schlüssel?
Öffentlicher Schlüssel?

Kann man ja mal verwechseln....

OpenID-Konfigurationen mit privaten Schlüsseln, mit Beispiel-Keys, sowie mit 512-Bit-RSA-Schlüsseln

Github Secret Scanning


$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 2.12 KiB | 2.12 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote: error: GH013: Repository rule violations found for refs/heads/main.
remote:
remote: - GITHUB PUSH PROTECTION
remote: ——————————————————————————
remote: Resolve the following violations before pushing again
remote:
remote: - Push cannot contain secrets
remote:
remote:
remote: (?) Learn how to resolve a blocked push
remote: https://docs.github.com/code-security/secret-scanning/working-with-secret-scanning-and-push-protection/working-with-push-protection-from-the-command-line#resolving-a-blocked-push
remote:
remote:
remote: —— GitHub SSH Private Key ——————————————————
remote: locations:
remote: - commit:
remote: path:
remote:
remote: (?) To push, remove secret from commit(s) or follow this URL to allow the secret.
remote: https://github.com/hannob/testtest/security/secret-scanning/unblock-secret/
remote:
remote:
remote:
To github.com:hannob/testtest.git
! [remote rejected] main -> main (push declined due to repository rule violations)
error: failed to push some refs to 'github.com:hannob/testtest.git'

Technische Details zu badkeys

Verschiedene Arten von Tests

Algorithmische Erkennung (Fermat, ROCA)

Liste von bekannten, unsicheren Schlüsseln (Debian-OpenSSL-Bug, Beispiel-Keys)

Hybrid (gemeinsame Primzahlen)

Liste von unsicheren Schlüsseln

Hash des öffentlichen Schlüssels?

Möglich, aber nicht optimal.

Öffentlicher RSA-Schlüssel N (Modulus), e (Exponent)

badkeys-Blockliste: Hash des Modulus N

Angenommen, wir haben zwei RSA Schlüssel mit identischem Modulus, aber unterschiedlichem Exponenten

Key 1: N_1, e_1 Key 2: N_1, e_2

Wenn ein privater Schlüssel bekannt ist, lässt sich der andere effizient berechnen

Beispiel: Debian-OpenSSL-Bug

Standardmäßig nutzt OpenSSL den Exponenten e=65537, aber eine Kommandozeilenoption erlaubt die Verwendung von e=3

Da badkeys nur den Modulus betrachtet, erkennt es derartige Keys automatisch

Elliptische Kurven

Es gibt verschiedene Möglichkeiten, öffentliche Schlüssel zu codieren (X/Y-Wert, X-Wert und Point Compression)

badkeys betrachtet bei elliptischen Kurven nur den X-wert und kann daher Details der Codierung ignorieren

badkeys in der Praxis

Demo Time!

badkeys installieren und vorbereiten

```
pip install badkeys
badkeys --update-bl-and-urls
```

badkeys Kommandozeile

Demo Time!

badkeys Erkennung

- Fermat-Faktorisierung
- Gemeinsame Primfaktoren
- ROCA
- Debian-OpenSSL-Bug
- keypair-Bug
- Zahlreiche öffentliche private Schlüssel

Open Source (MIT-Lizenz)

https://github.com/badkeys/badkeys

Zukunft

Ausweitung der erkannten kompromittierten Schlüssel

Regelmäßige Scans von DKIM, DNSSEC, TLS-Zertifikaten

(unterstützt von NGI0/NLnet)

Ideen? Sponsoring? Consulting?

Sprecht mich gerne an!

https://badkeys.info/

Fragen?