MFA-ing the Un-MFA-ble

Protecting Auth Systems' Core Secrets

👋 Hi, I’m Tal Be’ery

- Co-Founder, R&D @ ZenGo
- 20 years of cyber security experience
- Former EIR Innov8 VC, VP Research Aorato (acquired by Microsoft)
- @talbeerysec

👋 Hi, I’m Matan Hamilis

- Cryptography Research @ ZenGo
- 8 years of cybersecurity experience.

ZenGo

Founded
in 2017

VC backed
since 2018

30
employees

We're hiring!

Easy and Secure crypto experience:
all from your mobile device

Agenda

- SunBurst Incident: The role of persistence
- Golden SAML persistence attack
  - How SAML works
  - Golden SAML attack
- Solving Golden SAML
  - MFA (Multi-Factor Authentication) as reference
  - Current solutions and their limitations
  - Novel solution: Solving with modern crypto
- Distributed SAML: Threshold Signatures applied to SAML (+demo!)
- Takeaways + Q&A

SunBurst: Breach of the year

2020 United States federal government data breach

Incident

In 2020, a major cyberattack suspected to have been committed by a group backed by the Russian government penetrated thousands of organizations globally including multiple parts of the United States federal government, leading to a series of data breaches. Wikipedia

Target: U.S. federal government, state and local governments, and private sector

First reporter: FireEye (responsible disclosure); NSA (responsible disclosure); Reuters (public disclosure);

Duration: At least 8 months or 9 months

Suspects: Berserk Bear (Russia); Cozy Bear (Russia); FSB (Russia); SVR (Russia);

Location: United States

Scope of Russian Hacking Becomes Clear: Multiple U.S. Agencies Were Hit

The Pentagon, intelligence agencies, nuclear labs and Fortune 500 companies use software that was found to have been compromised by Russian hackers. The sweep of stolen data is still being assessed.

About 18,000 private and government users downloaded a tainted software update that gave Russian hackers a foothold into victims' systems, according to SolarWinds, the company whose software was compromised. Brendan Mcdermid/Reuters

By David E. Sanger, Nicole Perlroth and Eric Schmitt

Published Dec. 14, 2020 Updated May 10, 2021

Catalin Cimpanu

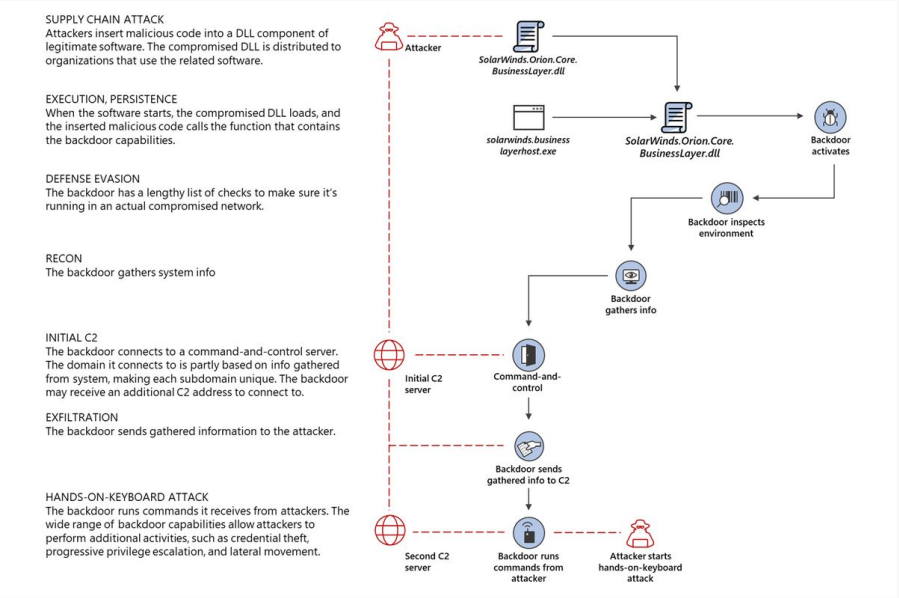
April 15, 2021

Government Nation-state News

White House formally blames Russian intelligence service SVR for SolarWinds hack

SunBurst APT

- Advanced Persistent Threat (APT):
  - Russian intelligence services
- Targets:
  - High profile US GOV agencies (+others)
- Most focus on “**Advanced**” initial access:
  - Supply chain compromise
  - Rogue version update to SolarWinds Orion to create a backdoor
- We want to focus on **“Persistent”**

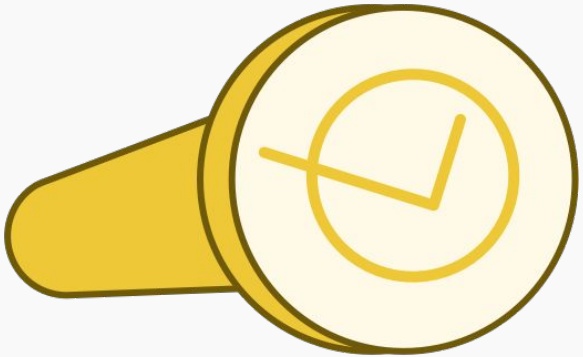


Persistence: APT vs. APT

- **Persistence is often a two way street**
- **Advanced Persistent Threats** often meet **Advanced Persistent Targets**
- Both sides are **advanced**
- Both side are **persistent**
  - Attackers are on a long term campaign
  - Defenders find attackers and clean the environment
  - No party has the luxury of doing something else
  - They must continue fighting each other
- The game is never over!

Persistence in practice

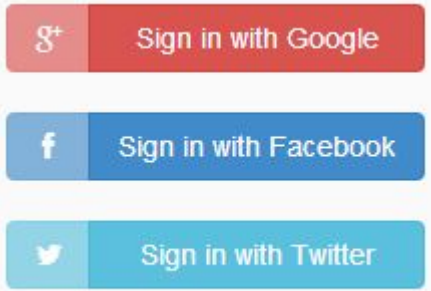
- MITRE ATT&CK tactic: “Persistence consists of techniques that adversaries use to keep access to systems”
- A popular way for attackers to maintain persistence is by targeting the targets’ **long term** secrets:
  - Single factor passwords
    - MFA mostly eliminates that, especially in APT targets
  - Keys used to generate access tokens:
    - Kerberos KRBTGT: “Golden Ticket”
    - SAML private key: “Golden SAML”



Golden SAML

What is SAML

- Modern corp environment is comprised of many web services, served by different vendors
- Each service has its own authentication solution
  - No SSO, many passwords to remember (or re-use), different MFA, users on-boarding / off-boarding / chane is a mess, etc.
- With SAML (Security Assertion Markup Language)
  - User management is removed from Service Providers (SPs) and centralized in Identity Provider (IdP)
- SAML analogies:
  - Corp version of “Sign in with …”
  - Web version of Kerberos

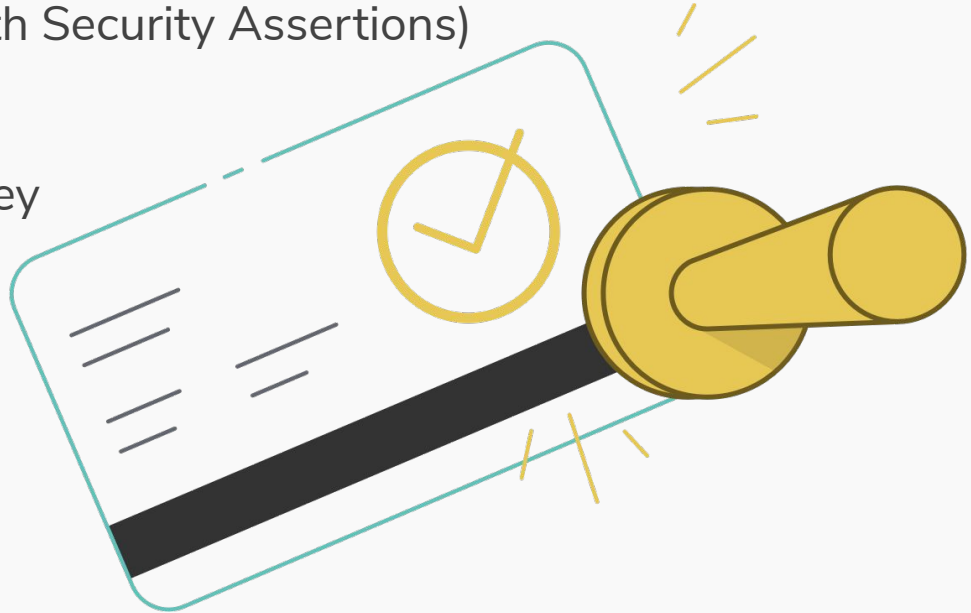


SAML logon step 1: Service Provider (SP)

- Users browses SP
- SP identifies that this user is using SAML
  - E.g. tal @ zengo.com
  - ZenGo configured SAML information beforehand
    - ZenGo SAML public key
- Sends user to IdP

SAML logon step 2: Identity Provider (IdP)

- Authenticates the user
  - Can use any Multi Factor Authentication (MFA)
  - Single-Sign-On (SSO)
- Generates a SAML token (XML with Security Assertions)
  - User's identity: email, name, etc.
  - User's attributes: e.g. admin / user
- Signs that SAML with its private key
- Send SAML token to user
- Sends user back to SP



SAML token example

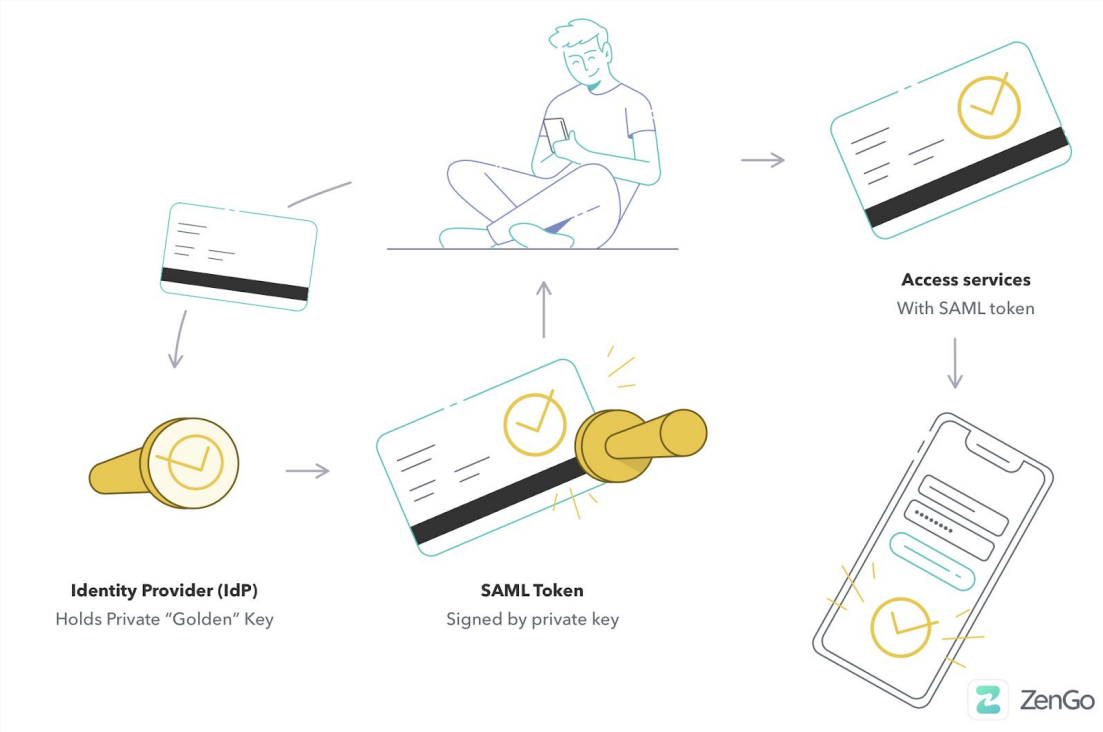
```
<ds:SignatureValue>MEUCIQD0T6u/kHShzHzbrL09GkW+znr3RGH4tISI/x5EYbL
    /awIgBZwYdGpfNPWbZubUSgNASnjhMFPKq740ZnCe6/d4D7Y=</ds:SignatureValue>
```

```
<saml:Attribute Name="eduPersonAffiliation" NameFormat="urn:oasis:names
    :tc:SAML:2.0:attrname-format:basic">
    <saml:AttributeValue xsi:type="xs:string">member</saml
        :AttributeValue>
    <saml:AttributeValue xsi:type="xs:string">student</saml
        :AttributeValue>
</saml:Attribute>
```

SAML logon step 3: Back to Service Provider

- When SP gets the IdP's signed SAML (via user)
  - Verifies the signature (with pre-configured public key)
  - Acts according to the security assertions

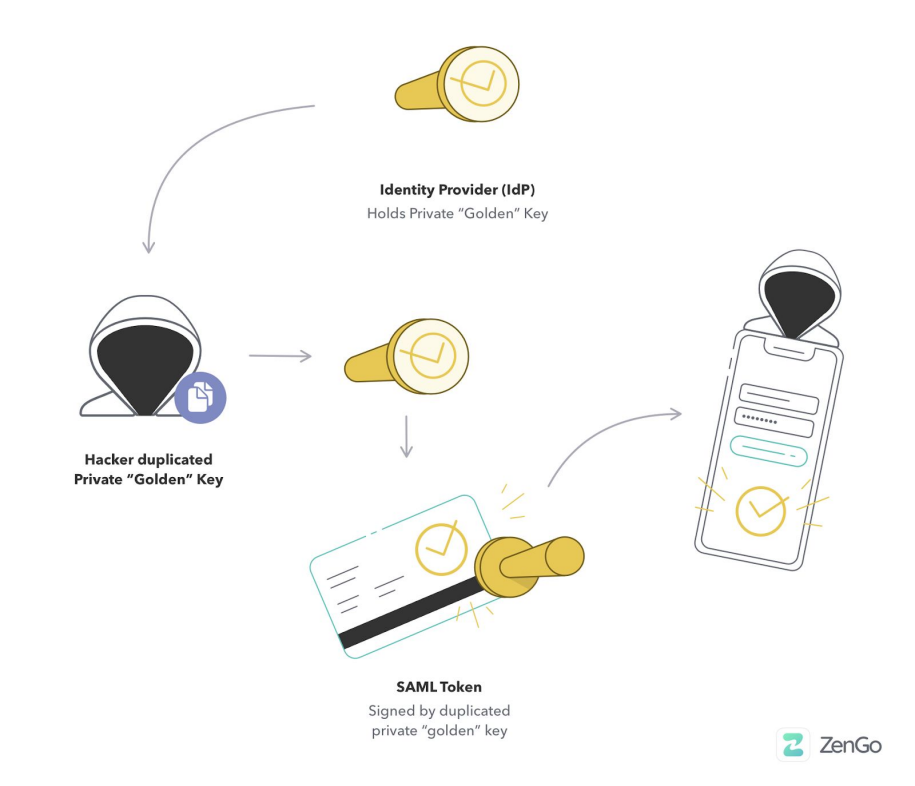
SAML flow: In high level



SAML is all about decoupling

- Authentication and Directory is decoupled from Service
  - Single Sign On
  - MFA
  - Users details in one place
    - Easy on-boarding / off-boarding
    - Updates to details
- SP and IdP do not talk directly, only through user
- The key pair is the only “glue”
  - IdP signs with private key
  - SP verifies with public key
- What happens if attackers steal private key?

Golden SAML: In high level



Golden SAML

- When attackers steal IdP's private key
- They become an alternative rogue IdP:
  - Can generate arbitrary access SAML tokens.
  - In an offline manner, within the attacker's environment
- Allow attackers to access all target's SPs
  - as any user
  - as any role
- Bypass original IdP security policies
  - Bypassing MFA
  - Bypassing access monitoring, if access is only monitored by IdP
- Golden SAML: coined by CyberArk in 2017 (@shakreiner)
- SunBurst: First publicly known use of the technique in the wild

Solving Golden SAML

Problem definition

- We want to solve Golden SAML, a persistence technique
- We want to solve the “offline” use of IdP private key
  - Attackers get a time limited access to IdP
  - Attackers get a long-term “offline” access to target’s assets
- Solving an “online” attacks on IdP is out of scope
  - Should be handled with our usual blue team methods against online, active attackers
    - XDR, process whitelist, etc.

MFA as a good solution reference

- MFA largely solved passwords as a persistence mechanism
  - “MFA can block over 99.9 percent of account compromise attacks” (Microsoft)
  - Bothers APTs enough to bypass them
- What makes MFA a good solution
  - **Composability**: Password is no longer a single point of failure
  - **Orthogonality**: The extra factors are actually different, i.e. not “2 passwords”
  - **Scalability**: we can add more factors if needed (SMS, retina, fingerprint, USB key)
  - **Short-lived**: The added factor value keeps rotating
- **Can we apply MFA principles to solve Golden SAML?**

Hardware based solution

- CISA advisory on ”Detecting Abuse of Authentication Mechanisms” recommends HSM (Hardware Security Module)

Strongly consider deploying a FIPS validated Hardware Security Module (HSM) to store on-premises token signing certificate private keys. An HSM, aggressively updated, makes it very difficult for actors who have compromised the system to steal the private keys and use them outside of the network [3].

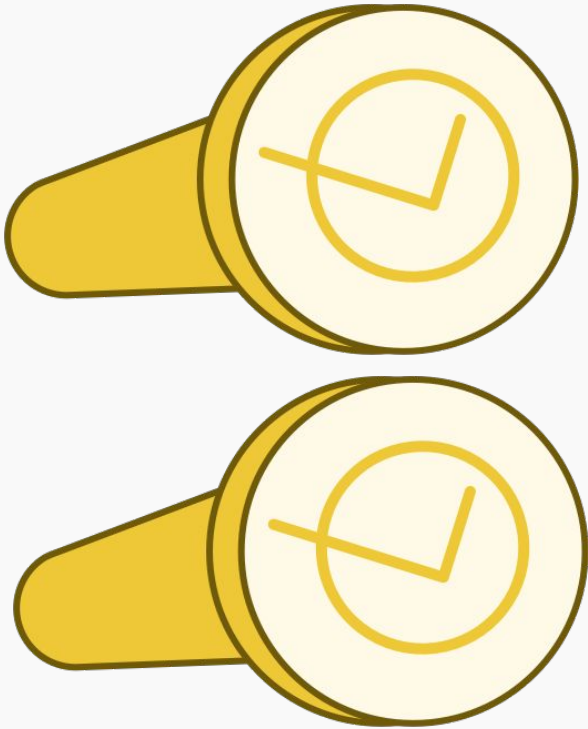
- In theory, HSM can sign yet prevent direct access to private key

HSM for SAML: Scorecard

- **Composability**: Private key is still a single point of failure
- **Orthogonality**: Does hardware residing in the same compromised environment provide enough resistance?
  - according to CISA only if it is “aggressively updated”
  - See Ledger’s BHUSA 19 talk on hacking HSM
- **Scalability:** Does not scale. We had gone from soft-ware to hard-ware, but what’s next? Harder-ware?
- **Short-lived:** Does not help with that

What if we can have multiple signers?

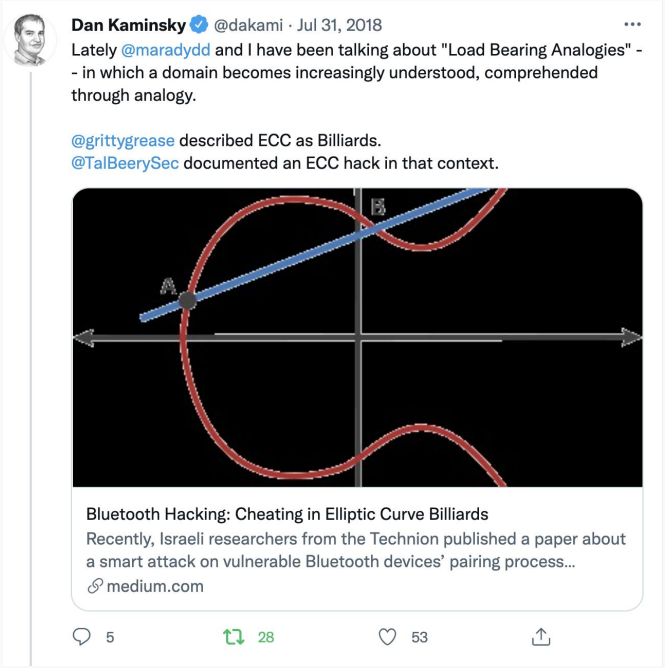
- Each token needs be signed by multiple parties
- These parties should be orthogonal
  - E.g. customer network and a 3rd party
- Success Criteria:
  - **Composability**: no single point of failure
  - **Orthogonality**: environments are orthogonal
  - **Scalability:** Scales
  - **Short-lived:** Still not so
- However this requires changes
  - IdP: that's relatively easy and interests are aligned
  - Standards and SPs
- Can we have multiple signers and change IdP only?



Threshold Signature Scheme (TSS)

- Modern cryptography magic
- Private key is created in a truly distributed manner
- Signing is done in a truly distributed manner
- Public key and signature verification remains the same,
  - Only signer (IdP) needs to be updated and nothing more (SPs)
- More reading on TSS for ECDSA
  - High level
    - Concepts
    - Use in blockchains
    - technical explanation
  - The papers Lindell 17, Genarro Goldfeder 18
  - ZenGo's TSS repository "Awesome TSS"

Tribute to Dan Kaminsky



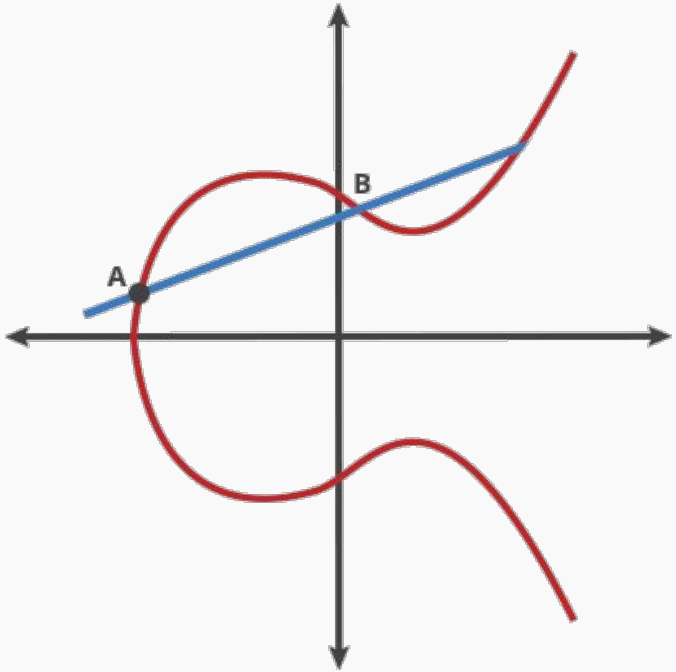
https://twitter.com/dakami

EC-DLP

- Every asymmetric crypto-system requires a hard problem
  - Hard to solve without the private key (Sk)
  - Easy to solve with the private key
  - Can be verified with public key (P)
- EC DLP: $P=Sk \times G$

EC-DLP as a billiards game

- Bizzaro billiards analogy (Nick Sullivan)
  - Addition in EC algebra is like a billiards' ball bouncing
- EC-DLP (P=Sk×G) is a Billiards game!
  - The ball is placed on point G
  - The ball is shot Sk times and ends on point P
  - No one can tell how many times the ball was shot (Sk)
  - Although they know start point (G) and end point (P)

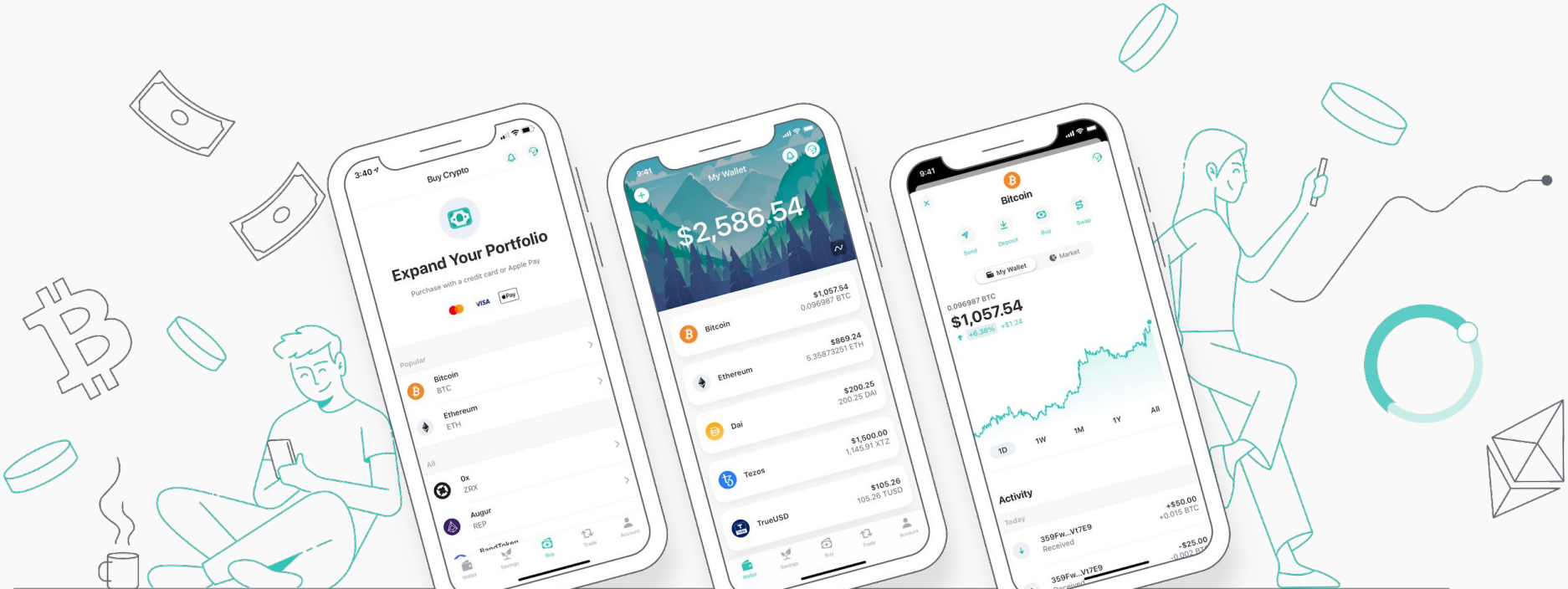


Distributed EC-DLP: Doubles' billiards game

- Bizzaro doubles' billiards analogy
  - The ball is placed on point G
  - The ball is shot Sk1 times and ends on point P1
    - $P1=Sk1 \times G$
  - No one can tell how many times the ball was shot (Sk1)
    - Although they know start point (G) and end point (P1)
  - If someone else now shoots Sk2 times from P1, EC-DLP is still a hard problem
    - $P= (Sk2) \times P1= (Sk2) \times (Sk1 \times G)=(Sk1 \cdot Sk2) \times G;$
- EC DLP is still hard with multiple players
  - $P=Sk \times G$
  - $P= (Sk1 \cdot Sk2) \times G;\ Sk = Sk1 \cdot Sk2$
- Additionally now the shares (Ski) can be rotated
  - $Sk = Sk1 \cdot Sk2 = (a \cdot Sk1)*(a\text{^}-1 \cdot Sk2)$

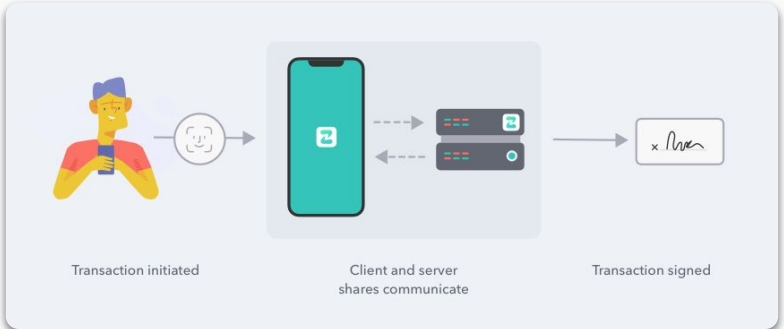
ZenGo makes crypto zen.

Buy, store, trade, and earn crypto in a tap.

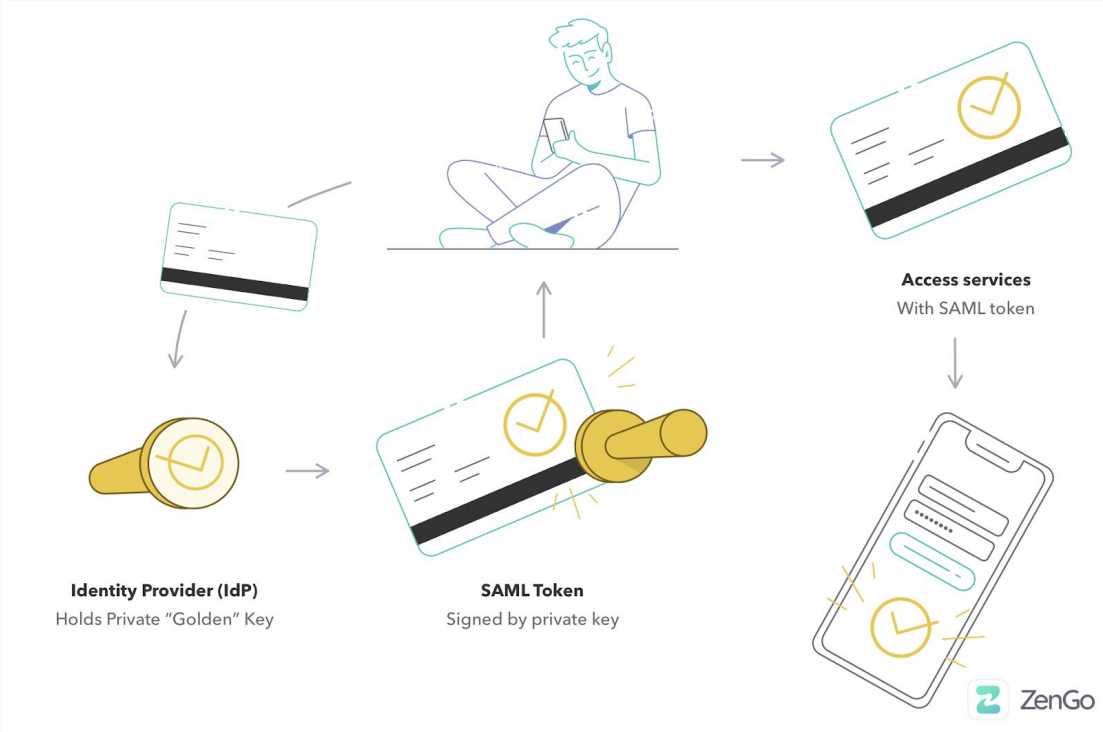


Threshold Signatures (TSS): 1 becomes 2

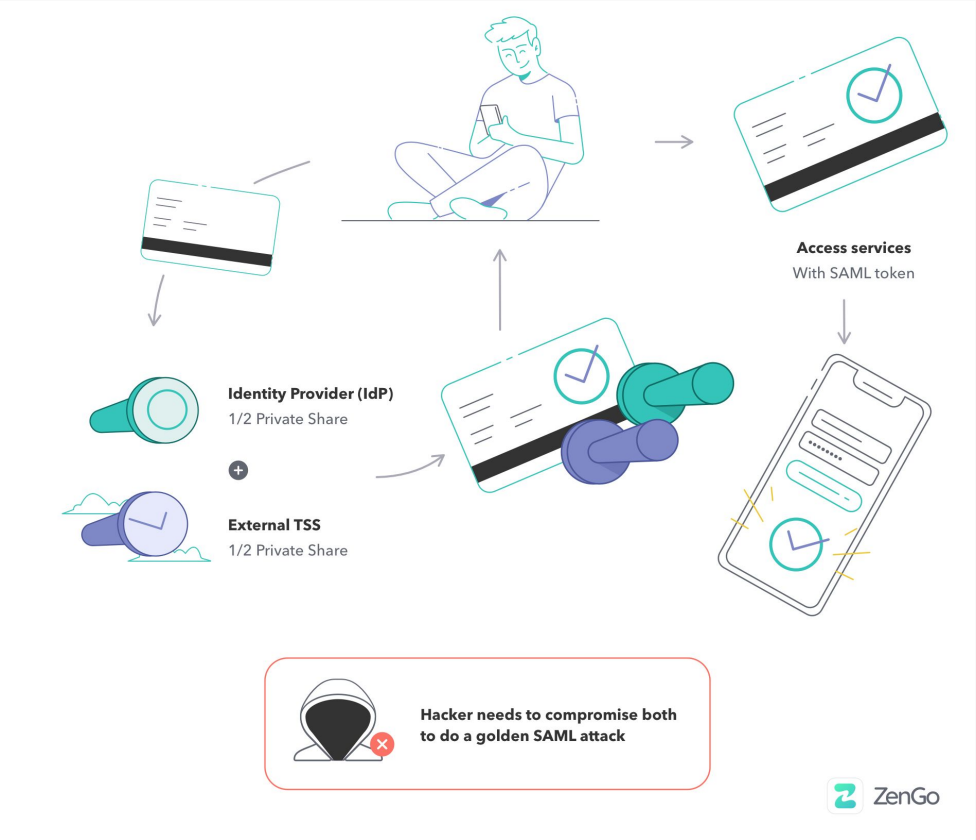
- Private key becomes distributed: no longer a Single-Point-of-Failure
- Distributed protocols: back and forth messages exchange between parties
  - Key generation: each party creates a “Share” (which is not “half of the key”)
  - Signing: using the Shares, parties sign together
- The signature looks the same!
- When **1 (private key) becomes 2 (shares)**:
  - Harder for attackers to steal: needs to compromise both parties
  - Easier to backup: each share is meaningless by itself



SAML flow: In high level



TSS SAML flow: In high level



TSS for SAML: Scorecard

- **Composability**: Private key becomes decentralized and no longer a single point of failure
- **Orthogonality**: Each share resides on a totally different environment
- **Scalability:** Number of parties is scalable. If 2 are not enough, why not 3? Or 4?
- **Short-lived:** Shares can be rotated without changing the main secret

TSS for SAML IdP

Demo Architecture

- Architecture is composed of:
  - Identity Provider.
  - Service Provider (agnostic of the TSS nature of the signature).
  - A multiparty TSS-ECDSA implementation.
- The code can be found at:
  - https://github.com/ZenGo-X/saml-demo

Demo Architecture

- IDP and SP by SimpleSAMLPhp.
  - Added support for the `http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256` algorithm for the `ds:SignatureAlgorithm` element at the IdP codebase.
  - The handler for this signature algorithm at the IdP calls for the multi-party signing routine.
  - The IdP and the SP will run in two different containers, each running the SimpleSAMLPhp codebase with the appropriate configuration.

Demo Architecture

- Multiparty TSS-ECDSA by ZenGo-X/multi-party-ecdsa.
- Demo signature scheme: 3-out-of-3.
- Signature algorithm used: Gennaro and Goldfeder - Fast Multiparty Threshold ECDSA with Fast Trustless Setup (AKA GG18).
- Each signer runs in a separate container.
- One of the containers is controlled by the IdP.
  - The rest are independent.
  - While at the demo all containers run on the same PC, a “real-world” implementation of this will comprise cosigners running within orthogonal, independent environments.
- The demo includes a distributed key generation (DKG).

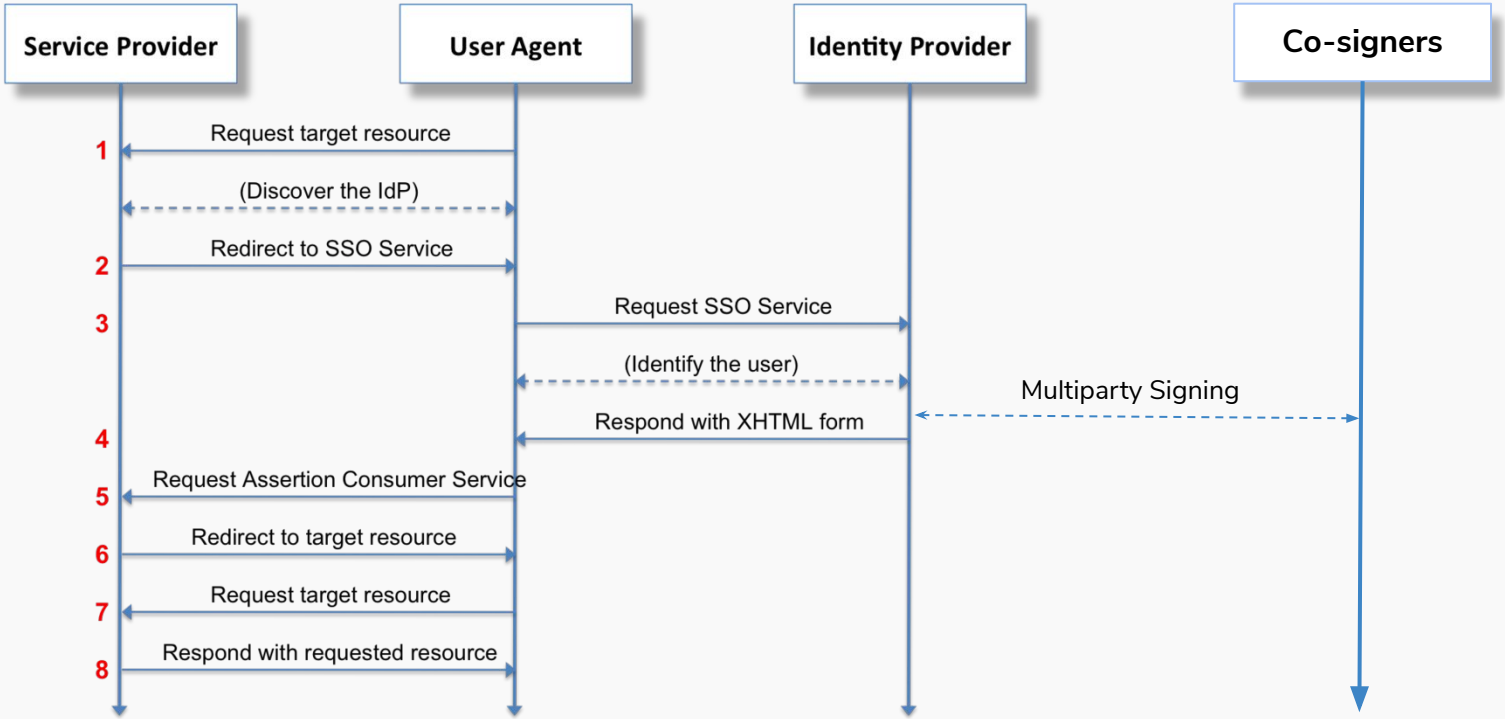
Demo Architecture - Setup Phase

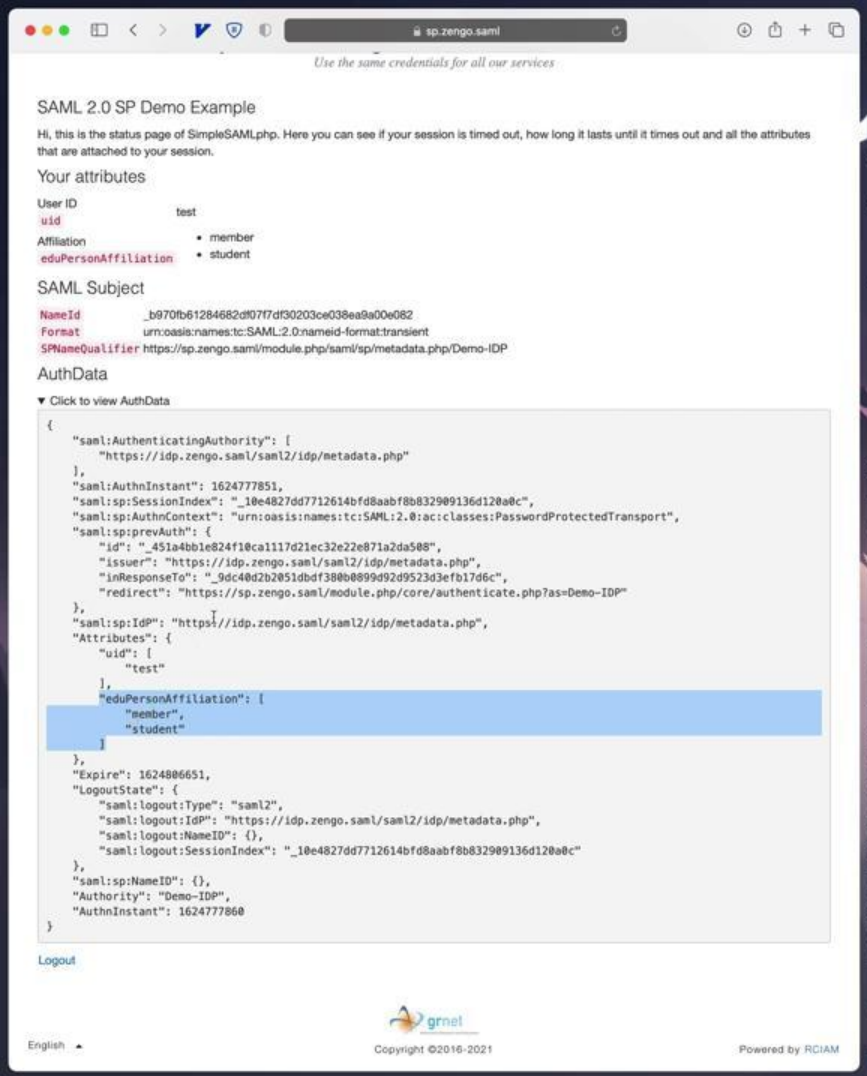
- Upon setup, we start all the signer containers.
- They run a DKG for a 3-out-of-3 multiparty ECDSA scheme.
- From the generated public key, we generate an X.509 certificate.
  - Private key isn't required to generate a self-signed certificate.
- This certificate is automatically passed to the SP.
  - The SP must hold a certificate of the IdP to verify the assertions.

Demo Architecture - Signing-in Phase

- When a user wishes to sign-in:
  - The SP redirects the client to the IdP.
  - The user fills-in a form and sends it to the IdP.
  - The IdP verifies the credentials.
  - If the verification succeeds, the IdP generates an unsigned assertion.
  - The assertion is sent to its controlled signer node.
  - The IdP control signing node initiates a signing session by sending its peers the assertion to be signed.
  - The containers cooperatively sign the assertion.
  - The signed assertion is sent back to the IdP.
  - The IdP redirects the client to the SP alongside its signed assertion.

Demo Architecture - Signing Phase

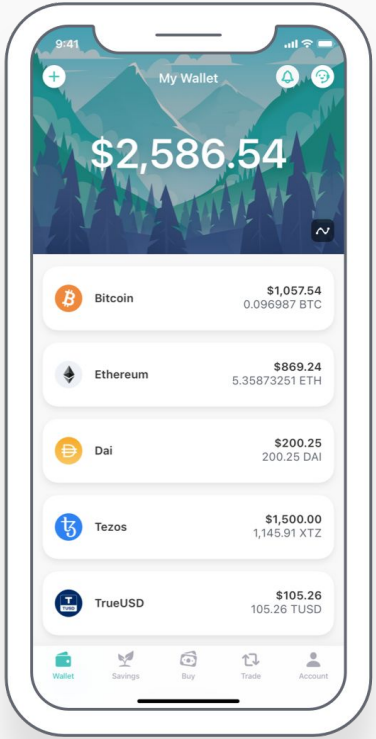



sp.zengo.saml
Use the same credentials for all our services
SAML 2.0 SP Demo Example
Hi, this is the status page of SimpleSAMLphp. Here you can see if your session is timed out, how long it lasts until it times out and all the attributes that are attached to your session.
Your attributes
User ID
uid
test
Affiliation
eduPersonAffiliation
member
student
SAML Subject
NameId _b970fb61284682df07f7df30203ce038ea9a00e082
Format urn:oasis:names:tc:SAML:2.0:nameid-format:transient
SPNameQualifier https://sp.zengo.saml/module.php/saml/sp/metadata.php/Demo-IDP
AuthData
▼ Click to view AuthData
{
"saml:AuthenticatingAuthority": [
"https://idp.zengo.saml/saml2/idp/metadata.php"
],
"saml:AuthnInstant": 1624777851,
"saml:sp:SessionIndex": "_10e4827dd7712614bfd8aabf8b832909136d120a0c",
"saml:sp:AuthnContext": "urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport",
"saml:sp:prevAuth": {
"id": "_451a4bb1e824f10ca1117d21ec32e22e871a2da508",
"issuer": "https://idp.zengo.saml/saml2/idp/metadata.php",
"inResponseTo": "_9dc40d2b2051dbdf380b0899d92d9523d3efb17d6c",
"redirect": "https://sp.zengo.saml/module.php/core/authenticate.php?as=Demo-IDP"
},
"saml:sp:IdP": "https://idp.zengo.saml/saml2/idp/metadata.php",
"Attributes": {
"uid": [
"test"
],
"eduPersonAffiliation": [
"member",
"student"
]
},
"Expire": 1624806651,
"LogoutState": {
"saml:logout:Type": "saml2",
"saml:logout:IdP": "https://idp.zengo.saml/saml2/idp/metadata.php",
"saml:logout:NameID": {},
"saml:logout:SessionIndex": "_10e4827dd7712614bfd8aabf8b832909136d120a0c"
},
"saml:sp:NameID": {},
"Authority": "Demo-IDP",
"AuthnInstant": 1624777860
}
Logout
English
grnet
Copyright ©2016-2021
Powered by RCIAM


Certificate:
Data:
Version: 3 (0×2)
Serial Number:
08:1e:4a:9e:33:15:18:5f:6b:aa:38:40:b0:3b:70
:91:19:0e:2b:14
Signature Algorithm: ecdsa-with-SHA256
Issuer: CN=ZenGo CA
Validity
Not Before: Jun 26 07:07:53 2021 GMT
Not After : Sep 25 07:07:53 2021 GMT
Subject: CN=idp.zengo.saml
Subject Public Key Info:
Public Key Algorithm: id-ecPublicKey
Public-Key: (256 bit)
pub:
04:d0:5f:cf:be:33:f9:04:c9:4a:e6:34:
99:ee:b8:
21:d7:a6:a0:ca:2c:a8:64:04:77:a0:af:
12:a8:8d:
9b:ea:1e:77:c9:12:83:a6:65:88:0a:af:
72:a5:b2:
8b:75:da:ac:81:68:6b:39:31:7d:5f:90:
34:3b:7e:
50:72:54:91:a3
ASN1 OID: secp256k1
Signature Algorithm: ecdsa-with-SHA256
30:45:02:21:00:a8:1c:79:df:8c:b4:1a:70:66:63:8a
:e7:e6:
a8:90:67:ae:64:f2:8d:1d:71:57:9a:8b:5c:eb:e5:b7
:33:ec:
e3:02:20:07:77:fc:09:38:74:c6:cd:b3:c3:3a:b9:60
:5a:cc:
e1:b9:35:0f:3d:4d:65:01:4c:4c:85:4a:ea:05:7a:9b
:14
-----BEGIN CERTIFICATE-----
MIIBKDCBz6ADAgECAhQIHkqeMxUYX2uqOECwO3CRGQ4rFDAKBggqhkjO
PQQDAjAT
MREwDwYDVQQDDAhaZW5HbyBDQTAeFw0yMTA2MjYwNzA3NTNaFw0yMTA5
MjUwNzA3
NTNaMBkxFzAVBgNVBAMMDmlkcC56ZW5nby5zYW1sMFYwEAYHKoZIzj0C
AQYFK4EE
AAoDQgAE0F/PvjP5BMlK5jSZ7rgh16agyiyoZAR3oK8SqI2b6h53yRKD
pmWICq9y
pbKLddqsgWhrOTF9X5A0O35QclSRozAKBggqhkjOPQQDAgNIADBFAiEA
qBx534y0
GnBmY4rn5qiQZ65k8o0dcVeai1zr5bcz7OMCIAd3/Ak4dMbNs8M6uWBa
zOG5NQ89
TWUBTEyFSuoFepsU
-----END CERTIFICATE-----
~/saml/samlphp/samlphp master* ↑ 23s
›


server.signer_1 | ++ printf %02x 32
server.signer_1 | + s_encoded=02201583c3a62758a3799cae3428d235b6b545828277f51c4536ddc3649a5dca9d4a
server.signer_1 | + elements_encoded=022031c637ec9de2f7ff7a897349b159febc4d2786c4c9b3f0925e6bdc61f4f533ba02201583c3a62758a3799cae3428d235b6b545828277f51c4536ddc3649a5dca9d4a
server.signer_1 | + elements_encoded_len=68
server.signer_1 | ++ printf %02x 68
server.signer_1 | + total_encoding=3044022031c637ec9de2f7ff7a897349b159febc4d2786c4c9b3f0925e6bdc61f4f533ba02201583c3a62758a3799cae3428d235b6b545828277f51c4536ddc3649a5dca9d4a
server.signer_1 | + echo -n 3044022031c637ec9de2f7ff7a897349b159febc4d2786c4c9b3f0925e6bdc61f4f533ba02201583c3a62758a3799cae3428d235b6b545828277f51c4536ddc3649a5dca9d4a
server.signer_1 | + nc -w 5 -q 0 -l 10000
server.signer_1 | + '[' 1 ']'
server.signer_1 | ++ nc -l 9000
3d3c2f64733a44696765737456616c75653e3c2f64733a5265666572656e63653e3c2f64733a5369676e6564496e666f3e
1_client.signer_1 | ++ xxd -r -p
1_client.signer_1 | ++ cut -f 1 -d ' '
1_client.signer_1 | ++ tr -d '\n'
1_client.signer_1 | ++ sha256sum
1_client.signer_1 | + MESSAGE=48afa01058d59e976c0a9a18a44df6b1b4a3abf9143d3153a6317d8ba1523369
1_client.signer_1 | ++ /mpe/target/release/examples/gg18_sign_client http://server.signer:8000 /keys.store 48afa01058d59e976c0a9a18a44df6b1b4a3abf9143d3153a6317d8ba1523369
1_client.signer_1 | ++ xargs
1_client.signer_1 | ++ sed -r -n 's/^.*SecretKey\((.*)\).*$/\1/p'
1_client.signer_1 | + OUTPUT='31c637ec9de2f7ff7a897349b159febc4d2786c4c9b3f0925e6bdc61f4f533ba 1583c3a62758a3799cae3428d235b6b545828277f51c4536ddc3649a5dca9d4a'
1_client.signer_1 | + '[' -n '' ']'
1_client.signer_1 | + '[' 1 ']'
1_client.signer_1 | ++ nc -l 9000
3d3c2f64733a44696765737456616c75653e3c2f64733a5265666572656e63653e3c2f64733a5369676e6564496e666f3e
2_client.signer_1 | ++ xxd -r -p
2_client.signer_1 | ++ sha256sum
2_client.signer_1 | ++ cut -f 1 -d ' '
2_client.signer_1 | ++ tr -d '\n'
2_client.signer_1 | + MESSAGE=48afa01058d59e976c0a9a18a44df6b1b4a3abf9143d3153a6317d8ba1523369
2_client.signer_1 | ++ /mpe/target/release/examples/gg18_sign_client http://server.signer:8000 /keys.store 48afa01058d59e976c0a9a18a44df6b1b4a3abf9143d3153a6317d8ba1523369
2_client.signer_1 | ++ sed -r -n 's/^.*SecretKey\((.*)\).*$/\1/p'
2_client.signer_1 | ++ xargs
2_client.signer_1 | + OUTPUT='31c637ec9de2f7ff7a897349b159febc4d2786c4c9b3f0925e6bdc61f4f533ba 1583c3a62758a3799cae3428d235b6b545828277f51c4536ddc3649a5dca9d4a'
2_client.signer_1 | + '[' -n '' ']'
2_client.signer_1 | + '[' 1 ']'
2_client.signer_1 | ++ nc -l 9000
0 1 SAML Demo *
2021-06-27 10:11 Matans-MacBook-Pr

Takeaways

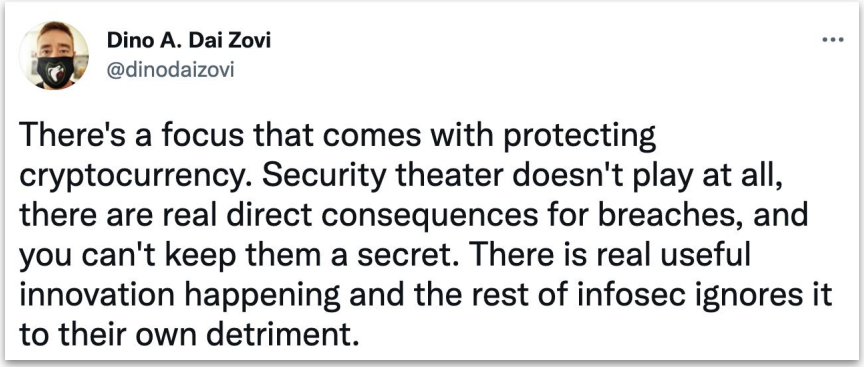


Takeaways

- APTs are targeting long term secrets for persistence
- Advanced Persistent Targets must solve
- Current hardware solutions are not perfect
- Using modern crypto (TSS) & relevant architecture enable better solutions

Generic Takeaways: Infosec ❤️ Cryptocurrency

- Cryptocurrency projects are solving hard security problems
- The Infosec community should embrace that



https://twitter.com/dinodaizovi/

twitter.com/zengo

medium.com/zengo

github.com/zengo-x

contact@zengo.com

ZenGo

www.zengo.com