

można przechwycić: mail nadawcy, odbiorcy, temat, treść wiadomości, imię usera, odpowiedzi serwera, hasło

protokół SMTP nie przewiduje szyfrowania normalnie, ale wersja z TLS (SMTPS) (Transport Layer Security) już tak

i tak, i nie - bo wiadomość na pewno jest zaszyfrowana i nie można przejąć jej snifferem, co nie znaczy, że gwarantuje to prywatność (ktoś przejmie konto, wyciek danych, coś na urządzeniu w stylu keylogger)

szyfrowanie wiadomości samemu - narzędzia inne np Pretty Good Privacy (zapewnia m.in. podpisy cyfrowe wiadomości, weryfikowanie nadawcy, szyfrowanie wiadomości) albo S/MIME (podobne do pgp)

korzystanie z protokołów z ssl/TLS

niedopuszczenie do przejęcia konta (2fa, dobre hasło)

proton: end-to-end encryption (mail jest zaszyfrowany na urządzeniu, przesłany w zaszyfrowanej formie, i odszyfrowany dopiero na kolejnym urządzeniu), password-protected email (umożliwia wysyłanie maili do osób spoza protona) zero-access encryption (proton nie może odczytać maili, są od razu szyfrowane na ich serwerach - proton nie może nikomu przekazać treści maila), wbudowany support do pgp, 2fa, kryptografia krzywych eliptycznych, wspiera Sender Policy Framework (aby nikt nie mógł wysłać maila z mojego adresu), Domain Key Identified Mail (że treść maili nie była zmieniana), DMARC (przeciw spoofowaniu adresu)

SPF - sender policy framework - określa, które serwery mają uprawnienia do wysyłania wiadomości w imieniu danej domeny.

DKIM - domain keys identified mail - podpisy cyfrowe, liczenie hasha z np nagłówków i treści

DMARC - domain based message authentication, reporting and conformance - dodanie rekordu dmarc do ustawień dns domeny z instrukcjami jak traktować maile

może się zdarzyć, że wiadomość trafi do spamu, gdy np zadziałają jakieś filtry antyspamowe - np gdy nadawca często wysyła wiadomości, to może zostać oznaczony jako spam; ogólnie mogą zadziałać różne algorytmy antyspamowe skrzynki odbiorcy, jakieś słowa kluczowe itp

może zawierać złośliwy załącznik, ponieważ nie zawartość załączników nie jest weryfikowana

DKIM pass, SPF nie - gdy mail jest git, ale idzie przez jakiś serwer, który nie jest wpisany

SPF pass, DKIM nie - jest wysłany z ok serwera, ale po jego opuszczeniu mail został jakoś zmodyfikowany

Użyć podpisu zaufanego można, gdy posiada się Profil Zaufany (trzeba mieć pesel)

z gov.pl:

> podpisać można dokumenty na dysku, jak i formularze elektroniczne

> podpisanie polega na kliknięciu "podpisz podpisem zaufanym" i potwierdzeniem kodem sms

> system generuje plik .xml z podpisem

pz jest bezpłatny, kwalifikowany nie

autentyczność pz zapewnia pieczęć elektroniczną ministra cyfryzacji. Zawiera dane np: pesel, identyfikator środka identyfikacji, czas. (podpisano w imieniu)

xades - XML Advanced Electronic Signature - format uniwersalny, podpisze się nim większość rzeczy, od pdf po jakieś jpg czy zip

weryfikacjapodpisu.pl

widac hash, format podpisu (xades czy pades), kto podpisal, kiedy, rodzaj uwierzytelnienia (Minister do spraw informatyzacji - pieczęć podpisu zaufanego), nazwe dokumentu info z profilu zaufanego: imie, nazwisko, pesel, id

informacje o wystawcy certyfikatu

"Certyfikat został zweryfikowany za pomocą:

weryfikacja podpisu elektronicznego Certyfikat nie znajduje się na liście CRL"

crl - certificate revocation list - lista unieważnionych certyfikatów - jest publikowana przez wystawce certyfikatów, zawiera numery seryjne tych unieważnionych (np przez ujawnienie klucza prywatnego); są dwa stany: wstrzymany, unieważniony

Ocsp - online certificate status protocol

Protokół zwracający informacje o statusie certyfikatu (poprawny, unieważniony, nieznany)

Wydanie zaświadczenia ze statusem „poprawny” oznacza, że certyfikat jest wystawiony przez jeden z podmiotów, których certyfikaty objęte są usługą OCSP oraz że podmiot ten na moment udzielania odpowiedzi skutecznie nie unieważnił sprawdzanego certyfikatu

po dostawieniu kropki krzyczy, ze integralnosc nie jest zachowana

podpis osobisty: jest to podpis z e-dowodu, ma taką samą moc jak podpis własnoręczny ALE tylko dla podmiotów publicznych, moze byc wykorzystywany w kontaktach z podmiotami innymi niż publiczne, ale tylko, jesli obie strony sie na to zgodza. podpisuje sie w eDO App

podpis zaufany: dwa formaty, podpisuje sie przez gov.pl, trzeba miec do tego profil zaufany, znaczek graficzny przy pdf, pz jest bezplatny, na okres trzy lat, trzeba przedluzac

podpis kwalifikowany: niczym podpis własnoręczny, posiadaczany za pomocą certyfikatu kwalifikowanego, mozna wgrac na nowy dowod, jest platny, mozna wybrac wydawcę (wolny rynek)

certificate authority can issue multiple certificates in the form of a tree structure. A root certificate is the top-most certificate of the tree, the private key which is used to "sign" other certificates. All certificates signed by the root certificate, with the "CA" field set to true, inherit the trustworthiness of the root certificate

plik .pem ( is a text-based container using base-64 encoding).

certy

x.509 - standard definiujacy format certyfikatu klucza publicznego

Certyfikat klucza publicznego zawiera trzy podstawowe informacje:

klucz publiczny podmiotu,

opis tożsamości podmiotu[3],

podpis cyfrowy złożony przez zaufaną trzecią stronę na dwóch powyższych strukturach.

Certyfikat klucza publicznego realizuje przede wszystkim funkcję autentyczności w stosunku do klucza publicznego podmiotu. Autentyczny klucz może być następnie używany do realizacji kolejnych funkcji bezpieczeństwa.

>

```
// key-scheduling algorithm do inicjalizacji permutacji tablicy S
void ksa(unsigned char *S, char *key) {
    size_t keylength = strlen(key);

    for (int i = 0; i < LENGTH; i++) {
        S[i] = i;
    }

    int j = 0;
    for (int i = 0; i < LENGTH; i++) {
        j = (j + S[i] + key[i % keylength]) % LENGTH;
        swap(&S[i], &S[j]);
    }
}

// pseudo-random generation algorithm
void rpga(unsigned char *S, char *input, unsigned char *output) {
    int i = 0;
    int j = 0;
    int index = 0;
    int k = 0;
    size_t len = strlen(input);

    for (int n = 0; n < len; n++) {
        i = (i + 1) % LENGTH;
        j = (S[i] + j) % LENGTH;
        swap(&S[i], &S[j]);
        index = (S[i] + S[j]) % LENGTH;
        k = S[index];
        output[n] = k ^ input[n]; // XOR
    }
}

void calculate_keys(struct key_pair *priv, struct key_pair *pub) {
    uint32_t n = prime1 * prime2;
    uint32_t euler = (prime1 - 1) * (prime2 - 1);
    uint32_t e = find_e(euler);
    uint32_t d = euclide(e, euler);
    priv->n = n;
    priv->key = d;
    pub->n = n;
    pub->key = e;
}
```

<p><b>OTP (ang. One-Time Password)</b></p> <ul style="list-style-type: none"> <li>■ rozwiążanie oparte o współdzielone klucze symetryczne</li> <li>■ implementowane w aplikacjach mobilnych/kluczach sprzętowych</li> <li>■ ogólny schemat generowania haseł jednorazowych:           <ul style="list-style-type: none"> <li>■ <b>klucz K, dodatkowe dane → algorytm OTP → hasło</b></li> </ul> </li> </ul>	<p><b>TOTP</b></p> <ul style="list-style-type: none"> <li>■ przy rejestracji/włączaniu usługi serwis generuje, zapisuje i przekazuje użytkownikowi <b>klucz K</b> (np. kod QR)</li> <li>■ użytkownik dodaje ten <b>klucz K</b> do aplikacji TOTP</li> <li>■ przy logowaniu do serwisu użytkownik wykorzystuje aplikację do wygenerowania hasła jako:           <ul style="list-style-type: none"> <li>■ <b>HMAC(K, czas)</b>, czas – aktualny zaokrąglony np. do pełnej minuty – hasło ważne przez 60s</li> <li>■ aplikacja wyświetla użytkownikowi odpowiednio zakodowane/przyjęte hasło (np. 6 cyfr)</li> </ul> </li> <li>■ użytkownik wprowadza hasło z aplikacji tam, gdzie się loguje</li> <li>■ po stronie serwera obliczane to samo i wynik porównywany</li> </ul>
<p><b>Rodzaje</b></p> <ul style="list-style-type: none"> <li>■ <b> HOTP</b> - oparty na HMAC (ang. HMAC-based One-Time Password)           <ul style="list-style-type: none"> <li>■ znormalizowany w ramach RF4226</li> <li>■ rolę dodatkowych danych odgrywa <b>licznik</b> (problemem utrata synchronizacji)</li> </ul> </li> <li>■ <b> TOTP</b> - oparty na czasie (ang. Time-based One-Time Password)           <ul style="list-style-type: none"> <li>■ znormalizowany w ramach RF6238</li> <li>■ rolę dodatkowych danych odgrywa <b>czas</b></li> <li>■ małe "oszustwo nazewnicze" – też korzysta z HMAC</li> </ul> </li> </ul>	<p><b>SSH (ang. Secure Shell Protocol)</b></p> <ul style="list-style-type: none"> <li>■ protokół sieciowy służący do bezpiecznej komunikacji zdalnej między dwoma komputerami</li> <li>■ ukierunkowany na zapewnienie szyfrowanej komunikacji dla usług sieciowych, w szczególności w przypadku zdalnego logowania</li> <li>■ RFC 4252<sup>a</sup> definiuje kilka metod uwierzytelniania           <ul style="list-style-type: none"> <li>■ "publickey" (jedyna wymagana)</li> <li>■ "password"</li> <li>■ "hostbased"</li> </ul> </li> <li>■ uwierzytelnianie za pomocą klucza możliwe po wygenerowaniu pary (klucz prywatny, klucz publiczny) i umieszczeniu klucza publicznego na serwerze           <ul style="list-style-type: none"> <li>■ dsa, ecsda, ed22519, rsa</li> </ul> </li> </ul>
<p><b>Cel:</b> Alicja (zna hasło) chce dokonać uwierzytelnionej wymiany klucza z serwerem</p> <p><b>Założenie:</b> Alicja zna klucz publiczny serwera</p>	<p><b>SSH (ang. Secure Shell Protocol)</b></p> <ul style="list-style-type: none"> <li>■ pewne dane charakterystyczne dla sesji są podpisywane przy użyciu klucza prywatnego użytkownika → podpis weryfikowany po stronie serwera</li> <li>■ Każdy, kto posiada klucz prywatny może się za nas uwierzytelnić, dlatego warto:           <ul style="list-style-type: none"> <li>■ stosować różne klucze do różnych serwerów (ograniczanie szkód przy wycieku)</li> <li>■ klucz prywatny przechowywać w formie zaszyfrowanej (domyślnie AES, wymagane hasło do odszyfrowania) \$ssh-keygen -t ed25519 i podajemy niepusty passphrase</li> </ul> </li> </ul> <p><b>Password-Authenticated Key Exchange (PAKE)</b></p> <p><b>wersje asymetryczne</b> (rozszerzone) - zapewniają uwierzytelnianie użytkownika w taki sposób, by nigdy nie musiał przekazywać hasła na serwera</p> <p><b>wersje symetryczne</b> - do innych celów, np. uwierzytelnianie przy dowodach osobistych, zakładają, że obie strony (dowód i czytnik) znają hasło (po jednej stronie hasło przechowywane a po drugiej użytkownik wprowadza)</p>
<p><b>Skąd pomysł?</b></p> <ol style="list-style-type: none"> <li>1 Jeżeli Alicja ma <math>(sk_A, pk_A)</math> to uwierzytelniona wymiana kluczy łatwa. Ale co jeśli Alicja nie ma jak bezpiecznie przechować kluczy? Zapamiętać może tylko hasło</li> <li>2 <math>sk_A = KDF(\text{hasło}, \text{soli})</math>. Sól utrudnia ataki. <math>pk_A</math> na serwerze. Ale jak spróbuję zalogować się w czymś imieniu, to dostanę sól z serwera, mogę offline policzyć potencjalne pary kluczy. Jak baza z publicznymi kluczami wycieknie, to sprawdzam</li> <li>3 Użyjmy OPRF do policzenia klucza z wykorzystaniem hasła (Alicji) i soli (serwera), żeby nie zdradzić ani hasła ani soli</li> <li>4 Jeżeli wykorzystamy 3. do policzenia klucza symetrycznego, możemy zaszyfrowany <math>sk_A</math> trzymać po stronie serwera</li> </ol>	<p><b>PACE</b></p> <ul style="list-style-type: none"> <li>■ dominujący protokół uwierzytelniania hasłem dla dokumentów potwierdzających tożsamość           <ul style="list-style-type: none"> <li>■ biometryczne paszporty i dowody osobiste w UE</li> </ul> </li> <li>■ celem zapewnienie, że używane są one wyłącznie za zgodą właściciela</li> <li>■ dość odpornie nie tylko na ataki biernego podsłuchu, ale także na aktywne ataki, gdzie przeciwnik może swobodnie wpływać na komunikację we wszystkich możliwych w praktyce sposobach</li> </ul>
<p><b>Rozwiążanie uważane za dojrzałe dla wiarygodnych urządzeń...</b></p> <p>okazuje się jednak podatne na <b>ataki kleptograficzne</b></p> <p><b>adwersarz</b> - może:</p> <ul style="list-style-type: none"> <li>■ wpływać na implementację protokołu PACE na urządzeniach,</li> <li>■ monitorować interakcję między urządzeniami użytkowników a czytnikami</li> </ul> <p><b>cele</b> - podsłuchiwanie komunikacji pomiędzy urządzeniem a uczciwym czytnikiem nie znając hasła użytkownika zapisanego w urządzeniu, złamanie hasła użytkownika π przykłady na tablicy</p>	<p><b>Podstawowe protokoły odbioru</b></p> <ul style="list-style-type: none"> <li>■ POP3 (ang. <i>Post Office Protocol version 3</i>), RFC 1939           <ul style="list-style-type: none"> <li>■ zorientowany na klienta (pobrane wiadomości standardowo usuwane z serwera)</li> <li>■ jeżeli pozostawione na serwerze, to przy ponownym połączeniu znów ściągane</li> <li>■ standardowo wykorzystuje port 110 (lub 995 dla TLS)</li> </ul> </li> <li>■ IMAP (ang. <i>Internet Message Access Protocol</i>), RFC 1733           <ul style="list-style-type: none"> <li>■ umożliwia zarządzanie wiadomościami na serwerze</li> <li>■ zachowuje synchronizację między różnymi urządzeniami dla których konto IMAP skonfigurowane</li> <li>■ po nawiązaniu połączenia z serwerem przesyłane są wyłącznie nagłówki wiadomości, po kliknięciu nagłówka pobierana jest cała wiadomość (online)</li> <li>■ po wybraniu odpowiedniej opcji można zsynchronizować wiadomości z serwera z lokalną skrzynką pocztową (offline)</li> <li>■ standardowo wykorzystuje port 143 (993 dla TLS)</li> </ul> </li> </ul> <p><b>Podstawowy protokół do wysyłania</b></p> <ul style="list-style-type: none"> <li>■ SMTP (ang. <i>Simple Mail Transfer Protocol</i>), RFC 821           <ul style="list-style-type: none"> <li>■ standardowo wykorzystuje port 25 (lub 465/587 dla TLS)</li> <li>■ protokół tekstowy, który definiuje reguły komunikacji między serwerami pocztowymi</li> </ul> </li> </ul>

## openpgp

### Wersja z podpisem (uwierzytelnienie wiadomości)

Alicja ma parę kluczy ( $x_A, y_A$ ), Boba ma parę kluczy ( $x_B, y_B$ )

- 1 Alicia liczy podpis pod wiadomością  $w$  jako  $s = \text{Sig}_{x_A}(H(w))$
- 2 Wiadomość jest kompresowana do  $m$ ,
- 3 Generowany jest  $K$  - losowy klucz symetryczny
- 4  $e = \text{Enc}(m||s, K)$  - skompresowana wiadomość i podpis są szyfrowane przy użyciu  $K$
- 5  $e_1 = E_y(K)$  - klucz  $K$  jest asymetrycznie szyfrowany przy użyciu  $y$  (jeżeli więcej odbiorców, to szyfrowany za pomocą klucza publicznego każdego z nich,  $e_1, \dots, e_n$ )
- 6 Zaszyfrowane wersje klucza  $K$  są konkatenowane z kluczem publicznym nadawcy  $y_A$  i zaszyfrowaną wiadomością i wysyłane do Boba (wszystkich odbiorców)

Po stronie odbiorcy (Boba) dochodzi jeszcze sprawdzenie podpisu

### Czym jest spoofing?

Technika polegająca na fałszowaniu lub modyfikowaniu informacji w celu podszcycia się pod inną osobę lub element systemu informatycznego

### Podstawowe rodzaje

- **DNS spoofing** – atakujący manipuluje odpowiedziami DNS, aby przekierować ruch sieciowy na fałszywe adresy IP (serwery kontrolowane przez atakującego)
- **caller ID spoofing** (fałszowanie informacji o dzwoniącym) – atakujący modyfikuje dane identyfikacyjne przekazywane odbiorcy podczas połączenia telefonicznego (fałszywe wyświetlanie numeru telefonu lub nazwy nadawcy na urządzeniu odbiorcy)
- **spoofing adresu e-mail** – atakujący modyfikuje adres nadawcy w nagłówku wiadomości e-mail w taki sposób, aby wyglądało to, jakby wiadomość pochodziła od innej osoby, organizacji lub źródła niż faktycznie

SPF →  
DKIM v

### Cel

Mogliwość wzbogacenia wiadomości o podpis z treści i różnych pól nagłówka (np. From). Serwer odbiorcy może zweryfikować autentyczność i integralność wiadomości wykorzystując odpowiedni klucz publiczny

### Co wykorzystuje?

Mechanizm DNS, który w rekordach danej domeny (lub subdomeny) przechowuje informację o kluczach publicznych

### Czym jest DMARC?

- mechanizm dopełniający dwa poprzednie
- zestaw reguł budowanych na bazie SPF i DKIM pozwalających zdecydować, czy wiadomość faktycznie przyszła od wysyłającego za którego się podaje
- adresuje problemy omówione przy SPF i DKIM a ponadto posiada możliwość raportowania wykrytych nieprawidłowości (SPF i DKIM mają charakter jednostronny, zwracając status fail nie są zobowiązane informować nadawcy o problemie)

### Wersja z podpisem (uwierzytelnienie wiadomości)

Alicja ma parę kluczy ( $x_A, y_A$ ), Boba ma parę kluczy ( $x_B, y_B$ )

- 1 Alicia liczy podpis pod wiadomością  $w$  jako  $s = \text{Sig}_{x_A}(H(w))$
- 2 Wiadomość jest kompresowana do  $m$ ,
- 3 Generowany jest  $K$  - losowy klucz symetryczny
- 4  $e = \text{Enc}(m||s, K)$  - skompresowana wiadomość i podpis są szyfrowane przy użyciu  $K$
- 5  $e_1 = E_y(K)$  - klucz  $K$  jest asymetrycznie szyfrowany przy użyciu  $y$  (jeżeli więcej odbiorców, to szyfrowany za pomocą klucza publicznego każdego z nich,  $e_1, \dots, e_n$ )
- 6 Zaszyfrowane wersje klucza  $K$  są konkatenowane z kluczem publicznym nadawcy  $y_A$  i zaszyfrowaną wiadomością i wysyłane do Boba (wszystkich odbiorców)

Po stronie odbiorcy (Boba) dochodzi jeszcze sprawdzenie podpisu

### Czym jest?

- jeżeli protokół zapewnia tajność w przód tzn., że ujawnienie długoterminowego klucza prywatnego i bieżącego sesyjnego nie daje możliwości deszyfrowania poprzednich sesji (o ile poprzednie klucze efemeryczne zapomniane)
- zwykle jest to związane z interaktywną wymianą klucza Diffiego-Hellmana
- protokół Signal zachowuje tajność w przód, ale inaczej

### Cel

Potwierdzenie, że dany serwer pocztowy mógł wysłać wiadomość e-mail w imieniu danej domeny

### Co wykorzystuje?

Mechanizm DNS, który w rekordach danej domeny (lub subdomeny) przechowuje informacje o serwerach pocztowych, uprawnionych do wysyłania wiadomości e-mail dla danej domeny

### Jak to działa?

Serwer pocztowy odbiorcy:

- odpytuje serwer DNS o wpis SPF dla domeny z pola MAIL FROM (RFC5321.MailFrom)
- jeśli serwer pocztowy nadawcy jest na liście dozwolonych serwerów, to otrzymujemy status pass

### Schemat podpisu (Gen, Sign, Verify)

$\text{Gen}(1^n) \rightarrow (\text{sk}, \text{pk})$  - algorytm generowania pary kluczy,  $\text{sk}$  - klucz prywatny (podpisujący) i odpowiadający mu  $\text{pk}$  - klucz publiczny (weryfikujący),  $n$  - parametr bezpieczeństwa

$\text{Sign}(\text{sk}, \text{m}) \rightarrow \sigma$  - algorytm podpisu wiadomości  $\text{m}$  przy użyciu klucza prywatnego  $\text{sk}$

$\text{Verify}(\text{pk}, \text{m}, \sigma) \rightarrow \{\text{true}, \text{false}\}$  - algorytm weryfikacji poprawności podpisu  $\sigma$ , pod wiadomością  $\text{m}$  przy użyciu klucza publicznego  $\text{pk}$

### Co zapewnia?

- uwierzytelnienie pochodzenia wiadomości - tylko właściciel klucza prywatnego  $\text{sk}$  odpowiadającego kluczowi publicznemu  $\text{pk}$  mógł podpisać wiadomość
- integralność (uwierzytelnienie) wiadomości - zmiana choć jednego bitu w podpisanej wiadomości powoduje, że podpis nie weryfikuje się poprawnie

## PPT-adversary

Adwersarz, którego moce obliczeniowe są modelowane przez probabilistyczny algorytm wielomianowy (ang. Probabilistic Polynomial Time)

## SIG — FORGE<sub>A,Π</sub>(n)

Eksperyment dla schematu podpisów  $\Pi = (\text{Gen}, \text{Sign}, \text{Verify})$ , adwersarza  $A$  typu PPT-adversary i parametru bezpieczeństwa  $n$ :

- wygeneruj tajny klucz  $sk$  i odpowiadający mu klucz publiczny  $pk$ , uruchamiając  $\text{Gen}(1^n)$
- adwersarz  $A$  otrzymuje klucz publiczny  $pk$  i dostęp do wyroczni podpisującej  $\text{Sign}(sk, \cdot)$ . Niech  $M$  oznacza zbiór wiadomości, dla których  $A$  poprosił o podpisy
- Wynikiem eksperymentu jest 1 wtedy i tylko wtedy, gdy:  $\text{Verify}(pk, m, \sigma) = 1$  oraz  $m \notin M$

## Generowanie kluczy

- wybierz dwie duże, różne liczby pierwsze  $p$  i  $q$  i oblicz ich iloczyn  $n = p \times q$
- oblicz  $\phi(n) = (p - 1) \times (q - 1)$ 
  - funkcja Eulera (toczent) - liczba reszt modulo  $n$  względnie pierwszych z  $n$
- wybierz losowo liczbę całkowitą  $e$  względnie pierwszą z  $\phi(n)$ , taką że  $1 < e < \phi(n)$
- oblicz (rozszerzony algorytm Euklidesa) odwrotność  $e$ , czyli  $d$  takie że  $e \times d \equiv 1 \pmod{\phi(n)}$
- klucz publiczny to  $pk = (n, e)$  a klucz prywatny to  $sk = (n, d)$

## Algorytm podpisu $\sigma = \text{Sig}(sk, m)$ , gdzie $sk = (n, d)$

- oblicz  $H(m)$  dla wiadomości  $m$ ,  $H : \{0, 1\}^* \rightarrow \mathbb{Z}_n$
- oblicz i zwróć  $\sigma = H(m)^d \pmod{n}$

## Problemy podstawowe

### 1 Książkowe RSA

- nie ma funkcji hashujących, założenie, że podpisywana wiadomość  $m \in \{0, 1, \dots, n - 1\}$
- **atak maskujący**: chcąc otrzymać podpis pod  $m$  dajemy komuś do podpisu  $r^e m \pmod{n}$ , ofiara liczy podpis  $\sigma = (r^e m)^d \pmod{n} = r m^d \pmod{n}$ , atakujący liczy podpis pod  $m$  jako  $\sigma' = \sigma / r$

### 2 Współdzielenie prywatnych wykładników lub modulo

- jeżeli podpisujący (atakujący) zna  $p, q$  takie, że  $n = pq$ , zna  $sk_1 = (n, d_1)$  i  $pk_1 = (n, e_1)$  i znajdzie kogoś, kto używa tego samego  $n$ , czyli  $pk_2 = (n, e_2)$ , to
  - może policzyć  $\phi(n) = (p - 1)(q - 1)$
  - za pomocą rozszerzonego algorytmu Euklidesa może policzyć  $d_2$  takie, że  $e_2 d_2 + y \phi(n) = \text{gcd}(e_2, \phi(n))$
- jeżeli atakujący ma tylko  $sk_1 = (n, d_1)$  i  $pk_1 = (n, e_1)$  i znajdzie kogoś, kto używa tego samego  $n$ , patrz algorytm na tablicy

## Definicja niepodrabialności

Schemat podpisów  $\Pi = (\text{Gen}, \text{Sign}, \text{Verify})$  jest niepodrabialny jeśli dla każdego probabilistycznego algorytmu czasu wielomianowego  $A$ , prawdopodobieństwo  $\Pr[\text{SIG} = \text{FORGE}_{A,\Pi}(n) = 1]$  jest zaniedbywalne

## Uwaga

W zależności od sposobu dostępu adwersarza do wyroczni podpisującą rozróżniamy niepodrabialność przy założeniu modelu ataku:

- z wyborem wiadomości (ang. chosen message attack) - adwersarz wybiera wiadomości, a następnie otrzymuje odpowiadające im podpisy. Celem jest stworzenie podpisu dla wiadomości, która dotąd nie została przedstawiona
- adaptacyjnego z wyborem wiadomości (ang. adaptive chosen message attack) - atak z wyborem wiadomości, gdzie wybór wiadomości może zależeć od otrzymanych dotychczas podpisów

## RSA

### Algorytm weryfikacji $\text{Verify}(pk, m, \sigma)$ , gdzie $pk = (n, e)$

- oblicz  $h = H(m)$  dla wiadomości  $m$  i funkcji hashującej  $H$
- oblicz  $h' = \sigma^e \pmod{n}$
- jeżeli  $h' = h$  zwróć true, w p.p. zwróć false

## Poprawność

$\sigma^e \pmod{n} \equiv (H(m)^d)^e \pmod{n} \equiv H(m)^{de} \pmod{n} \equiv H(m) \pmod{n}$ , ponieważ  $ed \equiv 1 \pmod{\phi(n)}$

## Uwaga

Znalezienie  $\phi(n)$  dla RSA modulo  $n$  jest jednoznaczne ze złamaniem RSA, bo mając  $\phi(n)$  i publiczne  $e$  możemy łatwo policzyć  $d$  (rozszerzony algorytm Euklidesa)

## Bezpieczeństwo

### 1 Trudność faktoryzacji dużych liczb

- faktoryzacja dużej liczby złożonej  $n$  na czynniki pierwsze jest znana jako trudny problem obliczeniowy, czyli mając  $p$  i  $q$  łatwo policzyć  $n = p \times q$ , ale mając tylko  $n$  trudno policzyć  $p$  i  $q$

### Trudności obliczeniowe w multiplikatywnej grupie modulo $n$

### Problem Dyskretnego Logarytmu dla grupy cyklicznej $\mathbb{G}$

Dla losowego elementu  $h \in \mathbb{G}$  i generatora  $g$  znajdź  $x$  takie, że  $g^x = h$  w grupie  $\mathbb{G}$

## Uwaga

Jeśli grupa  $\mathbb{G}$  jest grupą multiplikatywną modulo liczba pierwsza  $p$ , to problem logarytmu dyskretnego jest określany jako problem logarytmu dyskretnego w grupie  $\mathbb{G}$  modulo  $p$ . W tym przypadku, problem sprowadza się do znalezienia liczby całkowitej  $x$  takiej, że  $g^x = h \pmod{p}$

## ElGamal

### Verify( $y, m, \sigma$ ) - Algorytm weryfikacji podpisu $\sigma$ pod $m$

- sprawdź, że  $0 < r < p$ , jeżeli nie zwróć false,
- oblicz  $H(m)$
- oblicz  $v_1 = y^r r^s \pmod{p}$
- oblicz  $v_2 = g^{H(M)} \pmod{p}$
- jeżeli  $v_1 = v_2$  zwróć true, w p.p. zwróć false

### Uwaga: zakładamy, że przy składaniu podpisu i jego weryfikacji parametry $p, g, H$ są znane

## Dowód poprawności

Jeżeli podpis wygenerowany zgodnie z protokołem, to:

$$s \equiv (H(m) - xr)k^{-1} \pmod{p-1}$$

$$ks \equiv (H(m) - xr) \pmod{p-1}$$

$$H(m) \equiv (xr + ks) \pmod{p-1}$$

$$g^{H(m)} \equiv g^{(xr+ks)} \equiv (g^x)^r (g^k)^s \equiv y^r r^s \pmod{p}$$

## Generowanie kluczy

- wybierz dużą liczbę pierwszą  $p$
- wybierz generator  $g$  grupy multiplikatywnej  $\mathbb{Z}_p^*$
- wybierz tajny klucz prywatny  $0 < x < p - 1$
- Oblicz klucz publiczny  $y = g^x \pmod{p}$

## Algorytm podpisu $\sigma = \text{Sig}(x, m)$ pod wiadomością $m$

- wybierz losową liczbę  $0 < k < p - 1$ , względnie pierwszą z  $p - 1$
- oblicz  $r = g^k \pmod{p}$
- oblicz odwrotność  $k^{-1} \pmod{p-1}$  (rozszerzony algorytm Euklidesa)
- oblicz  $s = (H(m) - xr)k^{-1} \pmod{p-1}$ , gdzie  $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$  jest funkcją haszującą
- podpis pod wiadomością  $m$  stanowi para  $\sigma = (r, s)$

## Generowanie kluczy

- mając dane  $p, q, g$  dla grupy Schnorra wybierz losowy klucz prywatny  $0 < x < q$
- oblicz klucz publiczny  $y = g^x \bmod p$

$\text{Verify}(y, m, \sigma)$  - Algorytm weryfikacji podpisu  $\sigma$  pod  $m$

- oblicz  $v = g^s y^{-e}$
- oblicz  $e' = H(m||v)$
- jeżeli  $e' = e$  zwróć true w.p.p. zwróć false

**Uwaga:** zakładamy, że przy składaniu podpisu i jego weryfikacji parametry  $p, q, g, H$  są znane

## Dowód poprawności

Jeżeli podpis wygenerowany zgodnie z protokołem, to:  
 $v \equiv g^s y^{-e} \equiv g^s g^{-xe} \equiv g^k \equiv r \pmod{p}$   
Zatem  $e' = H(m||v) = H(m||r) = e$

## ECDSA

### DSA (ang. Digital Signature Algorithm)

- próba uniknięcia patentów na podpisy Schnorra
- propozycja amerykańskiego Narodowego Instytutu Standardyzacji i Technologii (NIST, ang. *National Institute of Standards and Technology*)
- standard DSS (ang. *Digital Signature Standard*) opisany w FIPS-186 (<https://nvlpubs.nist.gov/nistpubs/Legacy/FIPS/fipspub186.pdf>)
- bardziej klasyczna wersja ElGamala działająca w podgrupie multiplikatywnej rzędu  $q$  grupy  $\mathbb{Z}_p^*$ , z funkcją haszującą SHA
- DSA nie ma dowodów bezpieczeństwa takich jak dla Schnorra

- oparty na krzywych eliptycznych wariant DSA
- wyspecyfikowany w wielu normach (ISO 14888-3, ANSI X9.62, FIPS 186-2, IEEE P1363, ...), niestety nie wszystkie są kompatybilne
- nie ma dowodów bezpieczeństwa takich jak dla Schnorra, ale szeroko stosowany
- obie strony (podpisujący/weryfikujący) wiedzą, jakiej krzywej używają, znają jej rząd  $n$  - liczba punktów na krzywej oraz współpodzielne punktu bazowego  $G$
- podpisujący ma parę kluczy prywatny-publiczny  $(d, Q) = (d, dG)$ , gdzie  $d$  losowa liczba z zakresu  $[1, n - 1]$

## Podpisywanie

- oblicz  $h = \text{Hash}(m)$ , wykorzystane np. SHA-256, BLAKE2,  $h$  interpretowane jako liczba z przedziału  $[0, n - 1]$
- wybierz losowe  $k$  z zakresu  $[1, n - 1]$
- oblicz  $R = kG = (x_R, y_R)$
- oblicz  $r = x_R \bmod n$
- oblicz  $k^{-1} \bmod n$
- oblicz  $s = (h + rd)k^{-1} \bmod n$
- podpis stanowi para  $(r, s)$

## Veryfikacja $(r, s)$

- weryfikujący musi użyć tego samego  $h$
- oblicz  $s^{-1} \bmod n$
- oblicz  $u = hs^{-1} \bmod n$  oraz  $v = rs^{-1} \bmod n$
- oblicz  $R_1 = uG + vQ = (x_R, y_R)$
- jeżeli  $r = x_R \bmod n$  zwróć true w.p.p. zwróć false

## Porównanie RSA a ECDSA

`openssl speed ecdsap256 rsa4096`

- RSA - duży klucz prywatny, niewielki publiczny, ECDSA krótsze klucze
- ECDSA - krótsze podpisy
- podpisywanie ECDSA znacznie szybsze (169x)
- weryfikacja ECDSA nieco wolniejsza niż RSA

## EdDSA

### Własności

- algorytm podpisu oparty na krzywej Edwardsa
  - krzywa Edwards25519 (Ed25519), 128-bitowe zabezpieczenie
  - krzywa Edwards448 (Ed448), 224-bitowe zabezpieczenie
- nazwa myląca, schemat bazuje na podpisach Schnorra
- podpisy tworzone w sposób deterministyczny

### Infrastruktura Klucza Publicznego ang. *Public Key Infrastructure*

Zestaw zasad, procedur, technologii i usług zapewniających bezpieczne zarządzanie kluczami publicznymi wykorzystywanymi w kryptografii asymetrycznej  
(szyfrowanie, podpisy, uzgadnianie klucza DH...)

### Główne elementy

- 1 Certyfikaty cyfrowe – elektroniczne dokumenty, które potwierdzają tożsamość osoby/urządzenia/serwera  
 $\text{cert}_{CA \rightarrow A} = \text{Sign}_{sk_{CA}}(\text{"Klucz Alicji to } pk_A\text{"})$
- 2 Instytucje certyfikacyjne (ang. *Certification Authorities*) – **zaufane podmioty**, które wydają certyfikaty cyfrowe

## Pojedyncze CA

Najprostsza infrastruktura, ale uwaga:

- każdy kto chce polegać na usługach CA, musi uzyskać prawidłową kopię jego klucza publicznego  $pk_{CA}$  (bezpieczny, uwierzytelniony kanał?)
- przykład – jeżeli CA to komórka wewnętrz organizacji, to każdy pracownik może uzyskać  $pk_{CA}$  bezpośrednio w pierwszym dniu pracy
- pełne zaufanie przez wszystkich do pojedynczego punktu – mało praktyczne
  - wąskie gardło, ataki typu dos, czy odpowiada nam proces weryfikacji tożsamości tam stosowany, a co jeśli to CA zostanie skorumpowane, słabo chroni klucze prywatne do podpisu?

## Wiele CA

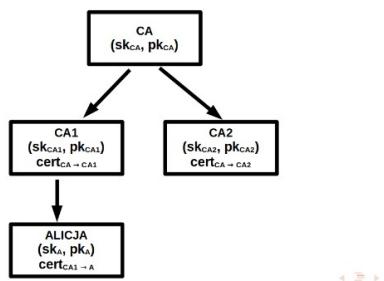
Alicja może wybrać kilka centrów certyfikacji i postarać się o kilka certyfikatów dla swojego klucza  $pk_A$  a każdy, kto otrzyma takie certyfikaty może wybrać, któremu ufa

- nie ma pojedynczego punktu awarii, ale ...
- bezpieczeństwo komunikacji strony weryfikującej certyfikaty jest tak dobre, jak najmniej bezpieczne CA, któremu ufa
- systemy operacyjne/przeglądarki internetowe zazwyczaj są już skonfigurowane z wieloma kluczami publicznymi CA, a domyślnym ustawieniem jest traktowanie wszystkich tych CA jako równie godnych zaufania
- czy trudno uzyskać miano CA?

## Przechodniość zaufania i łańcuchy CA

- zmniejsza część obciążenia dla pojedynczego CA
- nie rozwiązuje problemów związanych z bezpieczeństwem pojedynczego punktu awarii
- Alicja używając klucza  $pk_A$  dodaje:
 
$$cert_{CA1 \rightarrow A} = Sign_{sk_{CA1}}(\text{"Klucz Alicji to } pk_A\text{"}, data)$$

$$pk_{CA1}, cert_{CA \rightarrow A1} = Sign_{sk_{CA}}(\text{"Klucz CA1 to } pk_{CA1}\text{"})$$



## "web of trust"

- w pełni rozproszony, bez żadnego punktu centralnego
- każdy może wydawać certyfikaty dla innych,
- każdy musi samodzielnie zdecydować, jak bardzo ufać certyfikatom wydanym przez innych użytkowników
- wariant tego rozwiązania to PGP
- trudniej rozwiązać kwestię unieważniania certyfikatów

## Koncepcje bazowe

### ■ Wygaśnięcie

$$cert_{CA \rightarrow A} = Sign_{sk_{CA}}(\text{"Klucz Alicji to } pk_A\text{", data}),$$

gdzie  $data$  to pewna data z przyszłości, po której certyfikat staje się nieważny

- weryfikując ten certyfikat, trzeba znać nie tylko  $pk_A$ , ale i datę wygaśnięcia, i sprawdzić nie tylko czy podpis jest ważny, ale także czy data wygaśnięcia nie minęła
- użytkownik, który posiada certyfikat, musi skontaktować się z CA, aby wydał nowy, gdy ten wygaśnie (CA ponownie weryfikuje tożsamość)

## Koncepcje bazowe

### ■ Odwołanie

$$cert_{CA \rightarrow A} = Sign_{sk_{CA}}(\text{"Klucz Alicji to } pk_A\text{", sn}),$$

gdzie  $sn$  reprezentuje unikalny numer seryjny tego certyfikatu

- CA przechowuje informacje (Alicja,  $pk_A$ ,  $sn$ ) dla każdego wydanego certyfikatu
- jeśli  $sk_A$  odpowiadający  $pk_A$  zostanie skradziony, Alicja powiadamia CA (weryfikacja tożsamości Alicji)
- na koniec dnia CA generuje listę unieważnionych certyfikatów (CRL), a następnie podpisuje CRL i bieżącą datę
- weryfikacja certyfikatu wymaga sprawdzenia, czy podpis w certyfikacie jest ważny, numer seryjny nie występuje na najnowszej liście CRL oraz weryfikacji podpisu CA pod tą listą

- hierarchia - na szczeblu jest zaufane Root CA z certyfikatem podpisany przez siebie samego (ang. *self signed certificate*)
- dostajemy do weryfikacji nie jeden certyfikat, ale ich cały łańcuch (od certyfikatu dla EE (ang. *End Entity*) do Root CA)
- mamy transfer zaufania od góry do dołu hierarchii
- uwaga na okres ważności [Not Before, Not After] - dla węzła niższego w hierarchii okres ważności musi zawierać się w tym z węzła wyżej
- CA są odpowiedzialne za wskazywanie statusu unieważnienia certyfikatów, które wydają. Informacje o statusie unieważnienia mogą być udostępniane za pomocą:
  - protokołu Online Certificate Status Protocol (OCSP)
   
<https://datatracker.ietf.org/doc/html/rfc2560>,
  - list unieważnionych certyfikatów (CRL)
- w rozszerzeniach certyfikatu: punkty dystrybucji list CRL, dostęp do informacji o urządzie, gdzie zakodowany obiekt odpowiadający protokółu OCSP

## Podstawowa struktura

- TBS (ToBeSigned) Certificate – informacje, które podlegają podpisaniu przez wystawcę
  - wersja
  - numer seryjny certyfikatu
  - nazwa wystawcy certyfikatu (issuer)
  - nazwa podmiotu dla którego wystawiono (subject)
  - okres ważności certyfikatu
  - informacje o kluczu publicznym (algorytm, parametry, wartość klucza)
- Algorytm podpisu certyfikatu
- Podpis pod certyfikatem

## eIDAS

### Stworzony w celu

- opracowania wspólnych podstaw bezpieczeństwa interakcji elektronicznej między obywatelami, przedsiębiorstwami i organami publicznymi
- utworzenia infrastruktury klucza publicznego na poziomie ogólnoeuropejskim
- umożliwienia powstania wzajemnie uznawanych środków identyfikacji elektronicznej
- wzmocnienia współpracy krajów członkowskich w zakresie bezpieczeństwa i interoperacyjności poszczególnych systemów identyfikacji elektronicznej

### Dotyczy

- systemów identyfikacji elektronicznej (eID)
- usług zaufania

## Usługa zaufania

- tworzenie, weryfikacja i walidacja podpisów elektronicznych, pieczęci elektronicznych lub elektronicznych znaczników czasu, usług rejestrowanego doręczenia elektronicznego oraz certyfikatów powiązanych z tymi usługami
- tworzenie, weryfikacja i walidacja certyfikatów uwierzytelniania witryn internetowych
- konserwacja elektronicznych podpisów, pieczęci lub certyfikatów powiązanych z tymi usługami

Kryptologia (ang. *cryptology*) = κρυπτός (kryptós) + λόγος (lógos)  
ukryte słowo

- kryptografia (ang. *cryptography*) - tworzenie nowych szyfrów
- kryptoanaliza (ang. *cryptoanalysis*) - szukanie błędów bezpieczeństwa w istniejących i projektowanych szyfrach

tekst jawny (ang. *plaintext*) - wiadomość (tekst, grafika, audio...)

szyfrogram (ang. *ciphertext*) - tekst jawny po zaszyfrowaniu

klucz (ang. *key*) - zmieniona stanowiąca dodatkową informację wejściową

algorytm szyfrujący/deszyfrujący - algorytm pozwalający przekształcać tekst jawny na szyfrogram i na odwrót przy użyciu odpowiedniego klucza

## Szyfrowanie vs kodowanie

- kodowanie to tylko sposób zmiany reprezentacji wiadomości – kody ASCII, kodowanie UTF-8, kodowanie Base64
- zakodowany tekst każdy może odczytać, wystarczy wiedzieć, jaki algorytm został użyty
- do zaszyfrowania/odszyfrowania wiadomości potrzebna jest zaś dodatkowa zmienna – klucz

## Szyfry

- **symetryczne** - ten sam klucz do szyfrowania i deszyfrowania
- **asymetryczne** - klucz publiczny (jawny) do szyfrowania, odpowiadający prywatny (tajny) do deszyfrowania

## Szyfrowanie vs steganografia

- szyfrowanie ukrywa treść wiadomości, a nie sam fakt, że dwie strony się komunikują
- steganografia zajmuje się ukrywaniem istnienia komunikatu lub danych w innym medium w taki sposób, aby obserwatorzy nieświadomi istnienia komunikatu nie mogli go wykryć

## Uwagi

- szyfry asymetryczne są znacznie wolniejsze (obliczenia matematyczne na bardzo dużych liczbach vs operacje na bajtach/bitach)
- komputery kwantowe są większym zagrożeniem dla kryptografii asymetrycznej
- szyfrowanie symetryczne wygląda na lesze, ale jak obie strony mają ustalić wspólny klucz?

## Szyfrowanie hybrydowe

- 1 Nadawca szyfruje szyfrem symetrycznym tekst jawny przy użyciu jednorazowego, losowo wygenerowanego klucza  $K$  –  $Enc_K(m)$
- 2 Nadawca szyfruje klucz  $K$  szyfrem asymetrycznym z wykorzystaniem klucza publicznego odbiorcy –  $Enc'_{pk}(K)$
- 3 Nadawca wysyła do odbiorcy  $Enc_K(m)$  oraz  $Enc'_{pk}(K)$
- 4 Odbiorca deszyfruje swoim kluczem prywatnym  $sk$  (odpowiadającym kluczowi publicznemu  $pk$ ):  
 $Dec'_{sk}(Enc'_{pk}(K)) = K$
- 5 Mając klucz symetryczny  $K$  odbiorca deszyfruje  
 $Dec_K(Enc_K(m)) = m$  i dostaje tekst jawny  $m$

## Szyfry symetryczne

- **blokowe** – tekst jawny jest podzielony na bloki o ustalonej długości (np. 128b, 192b), każdy z nich jest szyfrowany osobno
- **strumieniowe** – tekst jawny  $M$  traktuje się jako strumień bitów, za pomocą ustalonego klucza  $K$  generuje się bity pseudolosowe, tzw. strumień klucza  $KS$  odpowiedniej długości i wówczas kryptogram stanowi:

$$KS \oplus M$$

## Szyfry strumieniowe

### Zasady ogólne

- są deterministyczne - bity pseudolosowe (strumień klucza) można na podstawie odpowiednich danych odtworzyć, co pozwala na jednoznaczne deszyfrowanie
- w pewnym sensie są podobne do PRNG (ang. *PseudoRandom Number Generator*)
- co do zasady algorytm  $SC$  generujący bity pseudolosowe (strumień klucza) powinien przyjmować dwie wartości:
  - tajny klucz  $K$
  - wartość jednorazową (oznaczana jako nonce  $N$  albo initial value  $IV$ ), która nie musi być tajna, ale powinna być niepowtarzalna dla każdego klucza

### Uwaga: brak uwierzytelniania!

- szyfr strumieniowy stanowy
- długo był najszerzej wykorzystywany szyfrem strumieniowym
- stan wewnętrzny to tabela  $S$  złożona z 256 bajtów, zainicjalizowana przy użyciu klucza  $K$  za pomocą KSA
- po wykonaniu KSA tabela  $S$  dalej zawiera wszystkie wartości bajtów 0...255, ale przepermutowane losowo, w zależności od klucza

### KSA (Key Scheduling Algorithm) dla klucza $K$

```
def KSA(K):  
    n = len(K)  
    j = 0  
    # tablica T - 256 elementow (powtarzane bajty klucza)  
    T = [K[i%n] for i in range(256)]  
    # S na tablica postaci S[0]=0, S[1]=1, ..., S[255]=255  
    S = list(range(256))  
    # generuj permutacje tablicy S na podstawie klucza  
    for i in range(256):  
        j = (j + S[i] + T[i]) % 256  
        S[i], S[j] = S[j], S[i]      #zamien S[i] z S[j]  
    return S
```

$$KS = SC(K, N)$$

$$C = M \oplus KS$$

$$M = C \oplus KS$$

- przy danym stanie początkowym  $S$  RC4 generuje strumień klucza KS długości  $m$  (liczba bajtów tekstu jawnego  $M$  do zaszyfrowania) zgodnie z poniższym algorytmem
- każda iteracja pętli for modyfikuje do 2 bajtów stanu wewnętrznego

### P-RGA (Pseudo-Random Generation Algorithm)

```
i = 0
j = 0
for b in range(m):
    i = (i + 1) % 256
    j = (j + S[i]) % 256
    S[i], S[j] = S[j], S[i]
    KS[b] = S[(S[i] + S[j]) % 256]
```

- na koniec  $C = KS \oplus M$

### Własności

- rodzaj szyfru symetrycznego
- tekst jawny podzielony na bloki o ustalonej długości (np. 128b, 192b)
- algorytm szyfrujący operuje na blokach tekstu jawnego i każdy z nich jest szyfrowany przy użyciu tego samego klucza
- deszyfrowanie również odbywa się na blokach danych o stałej długości

### Tryby pracy

- ECB (ang. *Electronic CodeBook*) – każdy blok przetwarzany niezależnie  $C_i = Enc_K(M_i)$
- CBC (ang. *Cipher Block Chaining*) –  $C_1 = Enc_K(M_1 \oplus IV)$ ,  $C_i = Enc_K(M_i \oplus C_{i-1})$ , dla  $i > 1$
- CTR (ang. *CounteR*) – przekształca szyfr blokowy w szyfr strumieniowy  $C_i = M_i \oplus Enc_K(N||Ctr + i)$

### Bezstanowość

HTTP to protokół **bezstanowy** - każda transakcja traktowana jako nowa

- mniejsze obciążenie serwera danymi
- kłopotliwe przy autoryzacji czy wielokrotnym korzystaniu ze strony/serwisu
- przydatne: ciasteczka, sesje, web storage

### Web Storage

- nowy mechanizm przechowywania par klucz-wartość wprowadzony przez HTML5
- wspierany przez większość nowoczesnych przeglądarek
- większy limit danych niż w przypadku ciasteczek (do 5MB)
- przechowywane dane nie są przesyłane na serwer
- 2 rodzaje (sessionStorage, localStorage)
  - ten sam interfejs (web storage API)
  - różna trwałość i zasięg danych
  - **localStorage** - ten sam dla danej aplikacji hostowanej na danym originie (protokół+domena+port), nieważne ile razy jest otwarta w danej przeglądarce - nawet w różnych oknach, dane przechowywane bez daty wygaśnięcia
  - **sessionStorage** - jest scisłe powiązany z sesją, czyli daną zakładką otwartą w przeglądarce
- nie zaleca się przechowywania danych wrażliwych (hasła, klucze) w tej pamięci, można np. info o postępie w grze

### Wykorzystanie protokołu TCP

- HTTP/1.1:**
- komunikacja polega na otwarciu jednego lub więcej połączeń TCP do serwera (w każdym połączeniu wysyłane pary żądanie-odpowiedź)
  - aby wysłać kolejne żądanie musimy poczekać na odpowiedź albo
  - obsługuje równolegle połączenia TCP do jednego serwera (kiedyś pipelining), każde nowe połączenie TCP "kosztuje" (3-way handshake, zasoby potrzebne na utrzymanie połączenia, nawiązanie TLS)
- HTTP/2:**
- komunikacja realizowana w ramach jednego połączenia TCP, którym przesyłane są ramki, ramki organizowane są w dwukierunkowe strumienie
  - wspiera multipleksowanie (możliwość jednoczesnego pobierania wielu zasobów)
  - serwer może nadawać poszczególnym zasobom priorytety

### TLS

- wykryto obciążenia statystyczne na RC4:
  - drugi bajt KS jest zerem z prawdopodobieństwem  $\frac{1}{128}$  (wyższe niż byśmy się spodziewali)
- później pojawiły się badania dotyczące obciążzeń statystycznych pierwszych 256 bajtów KS
- ataki w stylu rozgłaszanego - zakładają, że atakujący może uzbiereć dużo kryptogramów tej samej wiadomości zaszyfrowanej różnymi kluczami, celem odgadnięcia wiadomości (a nie złamanie klucza). Założmy, że chcemy odszyfrować pierwszy bajt wiadomości  $M_1$  mając:  $C_1^1 = M_1 \oplus KS_1^1$ ,  $C_1^2 = M_1 \oplus KS_1^2$ ,  $C_1^3 = M_1 \oplus KS_1^3$ ,  $C_1^4 = M_1 \oplus KS_1^4$  ...
- jeżeli pr. że bajty  $KS_1^i$  są zerem jest większe niż dla innych wartości, to jest bardziej prawdopodobne, że bajt  $C_1^i = M_1$
- atak polega na analizie milionów kryptogramów i szukaniu najczęściej powtarzającego się bajtu na danej pozycji

### Transakcja HTTP (HyperText Transfer Protocol)

- wysłanie żądania zasobu (Request) i oczekiwanie na odpowiedź
- odpowiedź serwera (Response) wraz z kodem odpowiedzi i odpowiednimi danymi

(domyślnie działa na porcie 80 w TCP)

### HTTPS (HyperText Transfer Protocol Secure)

HTTP + dane szyfrowane za pomocą **TLS** (kiedyś SSL)

- poufność przekazywanych danych
- integralność danych
- identyfikacja serwerów - certyfikacja (usługa płatna)
  - do celów akademickich można wykorzystać openssl, certbot

(domyślnie działa na porcie 443 w TCP)

### Ciasteczka

Ciasteczko jest to para (identyfikator, wartość) o określonej dacie ważności, przydzielana przez serwer do określonych zasobów. Dołączane są do żądania i mogą być ustawiane w odpowiedzi, przechowywane są po stronie przeglądarki. Rodzaje ciasteczek:

- stałe - ważne dłużej, umożliwiają zapamiętanie naszych preferencji
- sesyjne - przechowywane do zakończenia sesji przeglądarki
- podmiotów zewnętrznych - np. do personalizacji reklam

### Sesja

Sesja pozwala połączyć żądania HTTP z jednym użytkownikiem.

- dane przechowywane w pliku tymczasowym po stronie serwera,
- identyfikator sesji może być przekazywany:
  - 1 Za pomocą ciasteczka,
  - 2 Metodą GET jako argument w adresie strony.

### Różnice

- HTTP/1.1 stosuje format tekstowy komunikatów, wielokrotnie przesyła te same nagłówki w żądaniu/odpowiedzi
- HTTP/2 jest binarny - wiadomości mniejsze, odpowiedzi szybsze
- w HTTP/2 kompresja HPACK znacznie skracia rozmiary nagłówków żądania i odpowiedzi
- kompletnie zmieniono sposób wykorzystania protokołu transportowego TCP

### 5 podstawowych grup

- 100–199 – informacyjne z serwera
- 200–299 – żądanie przetworzone poprawnie
- 300–399 – żądanie przeadresowane; konieczne dalsze kroki
- 400–499 – żądanie niekompletne, błędne
- 500–599 – błąd w pracy serwera

## HTTP/3

- używa QUIC (Quick UDP Internet Connections) jako swojej warstwy transportowej, która jest zbudowana na UDP
- szybsze nawiązywanie połączenia dzięki zredukowanej liczbie wymiany komunikatów w porównaniu do TCP
- lepsze zarządzanie multipleksem połączonymi bez blokowania na pierwszym wierszu
- używa QPACK do kompresji nagłówków, zaprojektowanego do współpracy z multipleksem w QUIC
- lepsza wydajność dzięki szybszemu nawiązywaniu połączeń i lepszemu zarządzaniu utratą pakietów

## Bezpieczeństwo

- HTTP/3 jest zawsze szyfrowane, TLS 1.3 zintegrowane z QUIC
- HTTP/2 może działać na zwykłym TCP lub być szyfrowane za pomocą TLS

## HTTP/2

- jest dość skomplikowany - sama specyfikacja liczy ponad 100 stron
- ataki na protokół to głównie ataki typu DoS (patrz [https://www.imperva.com/docs/Imperva\\_H1\\_Http2.pdf](http://www.imperva.com/docs/Imperva_H1_Http2.pdf)), ale są różnej natury i wykorzystują różne podatności:
  - HPACK Bomb – atakujący przygotowuje duże żądanie http, które bardzo efektywnie się kompresuje, komunikacja niewielka, ale po rozpakowaniu wypełnia pamięć atakowanego serwera

## Co to jest?

Atak polegający na możliwości osadzenia w treści strony własnego kodu języka skryptowego (najczęściej JavaScript). Skrypt ten zostaje wykonany po stronie użytkownika w przeglądarce

## Kiedy pojawia się podatność?

Najczęściej, gdy w kodzie strony HTML pojawia się treść podana przez użytkownika

## Reflected XSS

Gdy złośliwy skrypt jest odbity z serwera jako część odpowiedzi HTTP. Oznacza to, że skrypt jest przesyłany do serwera jako część żądania HTTP (np. jako parametr URL, dane formularza), a następnie serwer w odpowiedzi zwraca ten skrypt, który jest wykonywany w przeglądarce ofiary. Najczęstszy przebieg:

- napastnik tworzy specjalnie przygotowany link zawierający złośliwy skrypt
- ofiara kliką ten link, wysyłając żądanie do serwera
- serwer nieświadomie włącza złośliwy skrypt w odpowiedź HTTP
- przeglądarka ofiary wykonuje złośliwy skrypt, który może np. kraść dane sesji, przechwytywać dane wejściowe użytkownika lub wykonywać inne niepożądane działania

## DOM-based XSS

Gdy złośliwy skrypt jest wstrzykiwany i wykonywany w kontekście Document Object Model (DOM) przeglądarki internetowej, a nie bezpośrednio przez serwer. Atak ten występuje, gdy aplikacja przetwarza dane wejściowe bezpośrednio w przeglądarce, modyfikując DOM w sposób, który może prowadzić do wykonania złośliwego kodu

- napastnik tworzy link zawierający złośliwy skrypt, który modyfikuje DOM strony internetowej
- napastnik wysyła link do ofiary
- ofiara kliką link
- skrypt przetwarza dane z URL i wprowadza zmiany w DOM strony. Jeśli te zmiany są wykonywane bez odpowiedniego sprawdzania, złośliwy skrypt może zostać wykonany

## Reflected XSS vs. DOM-based XSS

Główna różnica, przy DOM-based XSS całość ataku może odbyć się po stronie klienta

## Przykładowe kody odpowiedzi

- 200 OK – wszystko jest poprawne
- 201 Created – stworzono nowy adres URL przekazany w polu Location
- 300 Multiple Choices – podany adres prowadzi do więcej niż jednego zasobu
- 301 Moved Permanently – dokument nie znajduje się już pod podanym adresem, nowy adres w polu Location
- 400 Bad Request – błąd składniowy w żądaniu
- 401 Unauthorised – żądanie wymaga autoryzacji
- 404 Not Found – zasób nie został znaleziony
- 500 Internal Server Error – błąd w pracy serwera

## TLS1.2 vs TLS1.3

- Usunięcie przestarzałych funkcji
  - **TLS 1.2:** Obsługuje starsze i mniej bezpieczne algorytmy np. MD5, SHA-1, RC4, RSA Key Exchange
  - **TLS 1.3:** usunięto wsparcie dla przestarzałych algorytmów, korzysta z nowszych jak ChaCha20-Poly1305 i AES-GCM
- Tajność w przód (kompromitacja klucza prywatnego serwera nie wpływa na bezpieczeństwo wcześniejszych sesji)
  - **TLS 1.2:** Forward secrecy jest opcjonalna i zależy od wybranej metody wymiany kluczy (nie ma dla RSA Key Exchange)
  - **TLS 1.3:** Forward secrecy jest obowiązkowa, każda sesja wykorzystuje ephemeral Diffie-Hellman dla wymiany kluczy

- Prywatność: w TLS 1.2 SNI wysyłane są niezaszyfrowane

## Stored (persistent) XSS

Gdy złośliwy skrypt jest przechowywany bezpośrednio na serwerze (np. w bazie danych), a następnie jest dostarczany do wielu użytkowników. Ten rodzaj ataku jest szczególnie niebezpieczny, ponieważ złośliwy skrypt może zostać uruchomiony za każdym razem, gdy użytkownik/użytkownicy odwiedzą stronę, która zawiera zainfekowane dane

- napastnik umieszcza złośliwy skrypt w danych przechowywanych na serwerze, np. w formularzu komentarzy, w postach na forum
- serwer zapisuje te dane w swojej bazie danych lub innym trwałym magazynie danych bez odpowiedniej walidacji lub sanitizacji
- kiedy inny użytkownik przegląda stronę, która pobiera i wyświetla przechowywane dane, złośliwy skrypt jest wstrzykiwany do odpowiedzi HTML

przeglądarka użytkownika uruchamia złośliwy skrypt

## Metody obrony

- [https://cheatsheetseries.owasp.org/cheatsheets/Cross\\_Site\\_Scripting\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html)
- sanityzacja danych wejściowych
- enkodowanie danych w sposób zależny od kontekstu
- js: nie używać funkcji typu eval() przekazując do niej niezaufane niezwalidowane dane użytkownika
- js: nie używać funkcji i właściwości przypisujących bezpośrednio kod HTML do elementów drzewa DOM (np. innerHTML, document.write)
- wiele bibliotek i frameworków ma wbudowane mechanizmy zabezpieczeń przed atakami XSS
- nagłówki bezpieczeństwa HTTP, takie jak Content Security Policy (CSP) oraz nagłówek X-XSS-Protection, pomagają w zabezpieczaniu aplikacji przed atakami XSS, ograniczając możliwości wykonywania kodu JavaScript z niezaufanych źródeł

## Szyfrowanie dysków FDE (ang. *Full Disk Encryption*)

Szyfrowanie wszystkich danych, które są zapisywane na dysku i odszyfrowanie ich w momencie odczytu

- wszystkie dane szyfrowane jednym kluczem
- wprowadzenie poprawnego hasła/klucza powoduje, że dane dostępne do momentu zamknięcia systemu
- dostęp do danych mogą mieć wszyscy użytkownicy systemu
- dane mogą zostać wykradzione przez atakującego w momencie uzyskania dostępu do systemu

## Szyfrowanie na poziomie plików FBE (ang. *File-Based Encryption*)

Szyfrowanie mniejszych elementów danych (np. plików)

- każdy element może być zaszyfrowany innym kluczem
- dane mogą być odszyfrowane, kiedy są potrzebne
- użytkownicy dostają dostęp tylko do tych danych, które są dla nich przeznaczone

## Ataki

- rozwijały się adwersarza z fizycznym dostępem do dysku, przy założeniu jednej z trzech opcji: komputer włączony, komputer wyłączony i ofiara nieświadoma ataku, komputer wyłączony i ofiara świadoma ataku (np. skradziony dysk)
- wykryto sporo problemów, dotyczyły m.in tego, że nie ma kryptograficznego i fizycznego powiązania pomiędzy hasłem użytkownika a kluczem DEK(MEK) – można DEK odczytać bezpośrednio z układu wbudowanego w dysk (patrz np. slajd 93 prezentacji)

## Własności TPM

- odpowiedzialne za generowanie i przechowywanie kluczy kryptograficznych
- wspierają szyfrowanie/deszyfrowanie, weryfikację platformy programowo-sprzętowej pod kątem wprowadzonych w niej zmian
- posiadają sprzętowy generator liczb losowych
- posiadają moduł odpowiedzialny za generowanie hashy
- występują w komputerach najczęściej w postaci układu wbudowanego w płytę główną (np. Intel PTT, AMD fTPM), ale możliwe też podłączenie zewnętrznego modułu przez odpowiednie złącze na płycie

## Dyski HDD (Hard Disk Drive)

- najlepiej skorzystać z oprogramowania do bezpiecznego usuwania danych - zazwyczaj mają różne wzorce nadpisywania danych, np. Eraser (Windows):
  - jednokrotne nadpisanie danymi losowymi
  - jednokrotne nadpisanie samymi zerami (brytyj. HMG IS5 Baseline)
  - dwukrotne nadpisanie - zera + losowe dane (ros. GOST P50739-95)
  - trzykrotne przejście - zera + jedynki + losowe i weryfikacja zapisu danych (brytyj. HMG IS5 Enhanced)
  - trzykrotne przejście - losowe dane + losowy bajt + jego dopełnienie (US Army AR380-19)
  - trzykrotne przejście - zera + jedynki + dane losowe i weryfikacja po każdym przejściu (US DoD 5220.22-M(E))
  - siedmiokrotne przejście - zera (1,3,5) + jedynki(2,4,6) + dane losowe(7) i weryfikacja (kan. RCM TSSIT OPS-II)

## Niebezpieczeństwo przy SSD - proces garbage collection

Zeruje w tle te obszary pamięci, które zostały wcześniej oznaczone jako niewykorzystane. Operacja ta nie zawsze jest wykonywana niezwłocznie po oznaczeniu, co wskazuje na możliwość odczytania danych z tych obszarów do momentu jej zadziałania. W skrajnych przypadkach (np. błąd firmware'u) może nie zadziałać wcale. Wojskowe standardy nakazują fizyczne niszczenie dysków

## SED (ang. *Self-Encrypting Drives*)

Posiadają wbudowany moduł odpowiedzialny za szyfrowanie/deszyfrowanie danych

- dysk jest widoczny dla systemu operacyjnego jak każdy inny
- losowy 128(lub 256)-bitowy klucz DEK/MEK (Data/Media Encryption Key) jest generowany w momencie produkcji dysku
- wszystkie operacje (w tym szyfrowanie/deszyfrowanie) wykonywane w obrębie dysku (procesor komputera nie jest obciążany operacjami kryptograficznymi)
- funkcja *auto-lock* – klucz MEK przechowywany w postaci zaszyfrowanej –  $E_{KEK}(MEK)$ 
  - KEK (Key Encryption Key) generowany na podstawie wprowadzonego przez użytkownika hasła/PINu

## Secure Erase (Cryptographic erase)

Technika bezpiecznego usuwania informacji, która nie polega na fizycznym usunięciu danych a jedynie na wygenerowaniu i nadpisaniu klucza MEK do szyfrowania/deszyfrowania tych danych

nadpisujemy małą ilość danych  $\Rightarrow$  szybka operacja w momencie nie bezpieczeństwa przejęcia dysku

- każdy układ TPM ma w pamięci trwałe zapisane dwa rodzaje kluczy (generowana para kluczy RSA, prywatny zapisany):
  - klucz EK (ang. *Endorsement Key*) – generowane i zapisane w momencie produkcji (nie może być zmieniany)
  - klucz SRK (ang. *Storage Root Key*) – generowane i zapisane w trakcie inicjalizacji modułu (mogą być zmieniane przy ponownej inicjalizacji przez nowego właściciela)

## Dwie kategorie

- 1 **Stacked file system encryption** - szyfrowanie działające ponad systemem plików, gdy dane są przechowywane w zaszyfrowanych lokalizacjach a szyfrowanie działa w locie przed zapisaniem na dysk. Podczas odczytu folder z danymi jest odszyfrowywany na podstawie hasła podanego przez użytkownika
  - eCryptfs
  - EncFS
- 2 **Block device encryption** - szyfrowanie działające poniżej systemu plików, które zapewnia bezpieczeństwo wszystkich danych zapisanych na danym dysku przed próbą ich odczytania, ponieważ bez zamontowania i odszyfrowania na podstawie hasła użytkownika jest to jedynie ciąg losowo wyglądających danych bez informacji o systemie plików ani rodzaju danych
  - Loop-AES
  - VeraCrypt
  - dm-crypt (używany z LUKS<sup>a</sup>)

<sup>a</sup>Linux Unified Key Setup zapewnia kompatybilność pomiędzy różnymi dystrybucjami i zarządzaniem hasłami wielu użytkowników

## Dyski SSD (Solid State Drive)

- te dyski wykorzystują inny sposób przechowywania danych (zapisywanie na talerzach półprzewodnikowych a nie magnetycznych) i inaczej porządkują dane podczas zapisu
- wprowadzono w nich mechanizmy takiego rozdzielenia danych w pamięci, aby się równomiernie zużywały
- wielokrotne nadpisywanie danych jest tu nieefektywne i bardzo negatywnie wpływa na żywotność tych dysków
- przy zwykłym usuwaniu danych z dysku komórki oznaczane są tylko jako wolne, dane nie są usuwane
- dopiero komendą ATA Secure Erase do komórek są wysyłane krótkie skoki napięcia, co powoduje, że dane są zerowane
- najlepiej korzystać z Secure Erase z BIOS/UEFI
- można wykorzystać oprogramowanie do kasowania danych, np. GParted czy specjalistyczne dostarczane przez producentów dysków
- pod Linuxem ATA Secure Erase można wysyłać poprzez `hdparm`

<ul style="list-style-type: none"> <li>■ wytyczne dotyczące sposobów bezpiecznego usuwania danych z różnych typu nośników w zależności od stopnia ich poufności</li> <li>■ sposoby na usunięcie danych:           <ul style="list-style-type: none"> <li>■ <b>czyszczenie (ang. clear)</b> - nadpisywanie danymi niewrażliwymi, korzystanie ze standardowo dostępnych metod (np.przywracanie urządzenia do ustawień fabrycznych) uniemożliwiających dostęp do danych z poziomu interfejsu użytkownika</li> <li>■ <b>usuwanie (ang. purge)</b> - wykorzystanie oprogramowania lub sprzętu, który zapewni trwałe usunięcie danych, muszą to być techniki uniemożliwiające odzyskanie danych z wykorzystaniem metod laboratoryjnych</li> <li>■ <b>niszczenie (ang. destroy)</b> - techniki uniemożliwiające odzyskanie danych i ponowne użycie nośnika (rozmontowywanie, rozdrabnianie, topienie, palenie, rozdrabnianie dokumentów lub miękkich nośników w niszczarce i mieszanie)</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>■ przed przystąpieniem do niszczenia trzeba z dysku usunąć wszystkie oznaczenia, niszczenie komisyjne, spisuje się protokół</li> <li>■ czyszczenie - nadpisanie danych zerami co najmniej jednokrotne a przypadku pamięci flash co najmniej dwukrotnie</li> <li>■ cryptographic erase nie powinno być używane, jeżeli nie jest pewne, czy dane wrażliwe były obecne na dysku i mogły nie zostać usunięte przed włączeniem szyfrowania</li> <li>■ niszczenie dysków SSD - urządzenia powinny być w stanie rozdrobić na skrawki mniejsze niż 2mm, nie działa rozmagnesowywanie</li> </ul>
---	--