HW2 Rasoul_Salehi && Mojtaba_molaei

TLS/SSL.1

1.1.1 TLS/SSL Handshake Process: Key Exchange and Authentication

در ارتباط امن بین مرورگر و سرور که از TLS یا SSL استفاده می‌کنه، اولین کاری که انجام می‌شه، تبادل اطلاعات برای ساختن کلید رمزنگاری بین دو طرفه. این مرحله که بهش **Handshake** می‌گن، چند تا پیام اصلی داره که هر کدوم یه نقش مهم دارن:

---

1. ClientHello:

اول از همه، کلاینت (مثلاً مرورگر کاربر) یه پیام می‌فرسته به اسم ClientHello. توی این پیام اطلاعاتی مثل نسخه‌ی TLS که پشتیبانی می‌کنه، لیست الگوریتم‌های رمزنگاری که بلده، یه عدد تصادفی (که بعداً برای ساختن کلید استفاده می‌شه) و چند تا اطلاعات دیگه فرستاده می‌شه. اینطوری کلاینت نشون می‌ده که آماده‌ی شروع ارتباط امن هست.

---

2. ServerHello:

بعد سرور جواب می‌ده با ServerHello. توی این پیام، سرور یکی از الگوریتم‌های پیشنهادی کلاینت رو انتخاب می‌کنه و یه عدد تصادفی دیگه هم بهش اضافه می‌کنه. این عدد هم کنار عدد کلاینت استفاده می‌شه برای ساختن کلید نهایی.

---

3. Certificate:

حالا نوبت اینه که سرور هویتش رو ثابت کنه. برای این کار، گواهی دیجیتال (Certificate) خودش رو می‌فرسته. این گواهی شامل کلید عمومی سرور و اطلاعات امضاشده توسط یک مرجع معتبر (CA) هست. مرورگر می‌تونه با این اطلاعات مطمئن بشه که داره با یه سرور واقعی صحبت می‌کنه، نه یه نفر وسط راه.

---

4. ServerKeyExchange (اگه لازم باشه):

اگه سرور از الگوریتم‌های خاصی مثل Diffie-Hellman استفاده کنه، یه پیام جداگونه هم می‌فرسته که پارامترهای لازم برای این الگوریتم رو داره. اینجوری دو طرف می‌تونن یه کلید مشترک بسازن بدون اینکه کسی بتونه وسط کار شنود کنه.

---

5. ClientKeyExchange:

اینجا کلاینت یه اطلاعات مهم به اسم **pre-master secret** تولید می‌کنه و می‌فرسته برای سرور. این اطلاعات معمولاً با کلید عمومی سرور رمز می‌شن که فقط خود سرور می‌تونه بازش کنه. با کمک این pre-master secret و اون دوتا عدد تصادفی قبلی، کلاینت و سرور کلید اصلی رو می‌سازن.

---

6. ChangeCipherSpec + Finished:

بعدش دو طرف با هم هماهنگ می‌شن که از اینجا به بعد همه‌ی پیام‌ها رو رمز شده رد و بدل کنن. پیام Finished هم برای اینه که مطمئن بشن چیزی وسط راه دستکاری نشده.

در کل، این فرآیند باعث می‌شه دو طرف بدون اینکه کسی دیگه بتونه شنود کنه، یه کلید رمز مشترک داشته باشن که بعدش از اون برای رمز کردن اطلاعات استفاده می‌کنن.

1.1.2 Designing a Secure MAC + Encryption System

تو بخش Record Protocol در TLS، هدف اینه که هم **محرمانگی (confidentiality)** داده حفظ بشه و هم **درستی و صحت (integrity)** اون تضمین بشه. معمولاً توی این فرآیند، داده اول با یه الگوریتم رمزنگاری مثل AES رمز می‌شه، بعدش یه MAC یا همون کد احراز صحت (مثلاً با HMAC-SHA256) براش تولید می‌شه و به آخر پیام اضافه می‌شه.

ولی من می‌خوام یه روش ساده‌تر پیشنهاد بدم که هم محرمانگی رو تأمین کنه هم درستی رو، با استفاده از یه کلید از قبل به اشتراک گذاشته شده (Pre-Shared Key یا PSK).

طراحی روش ساده برای رمزگذاری و تضمین صحت:

۱. **فرض کنیم یه کلید مخفی بین دو طرف از قبل به اشتراک گذاشته شده باشه** (مثلاً یه کلید ۲۵۶ بیتی که از راه امن منتقل شده).

۲. وقتی می‌خوایم یه داده (مثلاً یک پیام متنی) رو ارسال کنیم:

- اول، اون داده رو با استفاده از یه الگوریتم متقارن (مثل AES در حالت GCM یا CBC) رمز می‌کنیم.
- بعد از رمز کردن، یه MAC تولید می‌کنیم، ولی نه از متن اصلی، بلکه از **متن رمز شده**. دلیلش اینه که اگه MAC رو از متن اصلی بگیریم، حمله‌گر می‌تونه از MAC هم برای درک محتوا سوءاستفاده کنه. ولی اگه از متن رمز شده بگیریم، این امکان از بین می‌ره.

۳. حالا پیام نهایی ما شامل دو بخشه:

- داده‌ی رمز شده
- و MAC تولید شده از داده‌ی رمز شده

۴. گیرنده وقتی پیام رو دریافت می‌کنه:

- اول، با استفاده از همون کلید مشترک، MAC رو دوباره محاسبه می‌کنه و با MAC دریافت‌شده مقایسه می‌کنه.
- اگه درست بود، یعنی داده در طول مسیر دستکاری نشده. بعدش، داده‌ی رمز شده رو با کلید مشترک رمزگشایی می‌کنه.

دلیل اینکه MAC رو از داده‌ی رمز شده می‌گیریم:

اگه MAC رو از متن اصلی بگیریم، حمله‌گر ممکنه با بررسی تطبیق‌ها و استفاده از اطلاعات آماری، چیزهایی از متن بفهمه. ولی وقتی از متن رمز شده MAC می‌گیریم، اطلاعات خیلی کمتری لو می‌ره و امنیت بیشتره.

با استفاده از یه کلید مخفی مشترک، توانستیم یه روش ساده بسازیم که:

- اول داده رو رمز کنه (محرمانگی)
- بعد ازش MAC بگیره (درستی)
- و در نهایت، گیرنده با بررسی MAC مطمئن بشه که داده دست‌کاری نشده.

این روش هم ساده‌ست، هم سریع، و مناسب برای ارتباط‌های سبک یا درون‌سازمانی.

1.1.3 Differences Between SSL 3.0, TLS 1.0, TLS 1.2

۱. تفاوت بین SSL 3.0 و TLS 1.0:

- این **TLS 1.0 در واقع نسخه‌ی به‌روزشده‌ی SSL 3.0 هست.** یعنی خیلی از ساختارهاش رو از اون گرفته، ولی باگ‌های جدی SSL 3.0 رو اصلاح کرده.
- در SSL 3.0، آسیب‌پذیری‌هایی وجود داشت که باعث شد حملاتی مثل POODLE بتونن اطلاعات رو در زمان رمزنگاری به دست بیارن.

-در TLS 1.0 الگوریتم‌های رمزنگاری جدیدتری داره، مثل HMAC که جایگزین MAC ساده در SSL 3.0 شد.

-و نیز TLS 1.0 پروتکل رمزنگاری رو طوری طراحی کرد که نسبت به SSL 3.0 قابل اطمینان‌تر باشه، مخصوصاً در مورد integrity داده.

---

۲. تفاوت بین TLS 1.0 و TLS 1.2:

- **استفاده از TLS 1.2 یه جهش خیلی بزرگ نسبت به TLS 1.0 بود.**
- توی TLS 1.0 فقط از الگوریتم‌های مشخص و قدیمی استفاده می‌شد، ولی در TLS 1.2 امکان استفاده از الگوریتم‌های جدیدتر (مثل SHA-256 و AES-GCM) اضافه شد.
- توی TLS 1.2، ساختار MAC و encryption به‌صورت کامل قابل تنظیم شدن. یعنی می‌شه الگوریتم‌های به‌روزتری رو انتخاب کرد که هم امن‌ترن، هم سریع‌تر.
- همچنین TLS 1.2 از حملاتی مثل BEAST و Lucky13 که روی نسخه‌های قبلی جواب می‌دادن، تا حد زیادی در امانه چون ساختار پردازش پیام تغییر کرده.
- یکی دیگه از مزیت‌هاش اینه که اجازه می‌ده الگوریتم هش توی امضای دیجیتال هم مشخص بشه. این خیلی مهمه چون تو نسخه‌های قبلی فقط SHA-1 استفاده می‌شد که دیگه امن نیست.

تفاوت‌های کلیدی بین نسخه‌های مختلف SSL و TLS

نسخه	ویژگی‌ها	مشکلات یا ضعف‌ها
SSL 3.0 (سال 1996)	نسبت به SSL 2.0 امن‌تر بود. از رمزنگاری متقارن، MAC و کلید عمومی استفاده می‌کرد.	آسیب‌پذیر به حمله POODLE. منسوخ شده و دیگه استفاده نمی‌شه.
TLS 1.0 (سال 1999)	جانشین SSL 3.0 با الگوریتم‌های رمزنگاری بهتر (مثل HMAC). ساختار (handshake) امن‌تر شد.	در برابر حملاتی مثل BEAST آسیب‌پذیره.
TLS 1.1 (سال 2006)	بهبود در مدیریت Initialization Vector (IV) برای جلوگیری از حمله BEAST.	هنوز از الگوریتم‌های رمزنگاری قدیمی استفاده می‌کرد. امنیت کافی نداشت.
TLS 1.2 (سال 2008)	اضافه شدن پشتیبانی از SHA-256، امکان انتخاب الگوریتم رمزنگاری سفارشی. پایداری و امنیت بالاتر.	همچنان نیاز به چند مرحله دست دادن داشت که باعث کندی ارتباط می‌شد.
TLS 1.3 (سال 2018)	حذف الگوریتم‌های ناامن مثل RC4، حذف RSA Key Exchange، کاهش تعداد round-trip در handshake. فقط الگوریتم‌های قوی باقی موندن.	امن‌ترین و سریع‌ترین نسخه تا به امروز.

بهبودهای مهم در TLS 1.3 نسبت به نسخه‌های قبلی

1. **حذف الگوریتم‌های ضعیف و ناامن:**
   - الگوریتم‌هایی مثل RC4، SHA-1، MD5، و حتی RSA برای key exchange حذف شدن.
   - این کار باعث شد حملات کلاسیکی مثل MITM، downgrade و key recovery بی‌اثر بشن.
2. **ساده‌سازی handshake:**
   - در TLS 1.2، راه‌اندازی ارتباط معمولاً دو round-trip لازم داشت، ولی در TLS 1.3 فقط **یک round-trip** نیاز هست.
   - این یعنی سرعت راه‌اندازی کانال امن خیلی بالا رفته، که مخصوصاً برای موبایل و اینترنت‌های با تأخیر زیاد خیلی مفیده.
3. **پشتیبانی از Forward Secrecy به‌صورت اجباری:**
   - فقط الگوریتم‌های تبادل کلید به صورت Ephemeral (مثل Diffie-Hellman موقتی) مجاز هستن.
   - این یعنی اگه حتی کلید خصوصی لو بره، ارتباطات گذشته همچنان امن می‌مونن.
4. **رمزنگاری داده‌ها سریع‌تر و امن‌تر شده:**
   - استفاده از AES-GCM و ChaCha20-Poly1305 برای رمزنگاری امن و سریع در دستگاه‌های مختلف.
   - این باعث شده هم امنیت بهتر بشه، هم کارایی توی سیستم‌هایی با سخت‌افزار ضعیف‌تر حفظ بشه.

تأثیر این بهبودها بر امنیت و کارایی:

معیار	قبل از TLS 1.3	در TLS 1.3
سرعت اتصال	کندتر (2 مرحله handshake)	سریع‌تر (1 مرحله handshake)
امنیت	امکان حملات downgrade و استفاده از الگوریتم ضعیف	حذف کامل الگوریتم‌های ناامن
پشتیبانی از Forward Secrecy	اختیاری بود	اجباری و پیش‌فرض
حجم کد اجرایی	پیچیده و بزرگ	ساده‌تر و کوچکتر

درکل TLS 1.3 به خاطر حذف الگوریتم‌های ضعیف، کاهش پیچیدگی handshake و استفاده از الگوریتم‌های مدرن، هم امنیت ارتباطات رو بالا برده و هم باعث شده تبادل اطلاعات سریع‌تر و کم‌هزینه‌تر انجام بشه. امروزه تقریباً همه‌ی مرورگرهای مدرن و وب‌سرورها به سمت استفاده از TLS 1.3 رفتن چون بهینه‌ترین نسخه محسوب میشه.

1.2.1 Main Difference: OCSP vs CRL

سوال : CRL (Certificate Revocation List) چیست؟

در اصل CRL یه فایل یا لیست هست که توسط صادرکننده‌ی گواهی (CA) منتشر می‌شه و داخلش شماره سریال گواهی‌هایی که لغو (revoke) شدن قرار داره. مرورگر یا کلاینت برای چک کردن اعتبار یه گواهی باید این فایل رو از سرور CA دانلود کنه و ببینه آیا گواهی مورد نظر توی اون لیست هست یا نه.

مشکل CRL چیه؟

- **سنگینی فایل:** اگه تعداد زیادی گواهی لغو شده باشه، لیست حجیم می‌شه و هر بار دانلودش زمان‌بر و پرهزینه‌ست.
- **بروزرسانی کند:** گاهی فایل CRL ساعت‌ها یا حتی یک روز دیرتر از لغو گواهی آپدیت می‌شه، یعنی ممکنه کلاینت متوجه لغو یه گواهی مهم نشه.

- **بار زیاد روی سرور CA:** همه‌ی کلاینت‌ها باید این فایل بزرگ رو بگیرن، که فشار زیادی به سرور میاره.

سوال OCSP (Online Certificate Status Protocol) چیست؟

درواقع OCSP یه روش سبک‌تر و به‌روزتره که به جای دانلود کل لیست، کلاینت فقط درباره‌ی یه گواهی خاص از سرور OCSP سؤال می‌پرسه:
"آیا این گواهی هنوز معتبره؟"
و سرور هم پاسخ می‌ده: **"good" (معتبره)، "revoked" (لغو شده)، یا "unknown".**

مزیت OCSP نسبت به CRL:

- **در لحظه جواب می‌ده** (real-time).
- **فقط وضعیت یه گواهی خاص رو می‌پرسه.**
- **سرعت بالاتر، حجم ترافیک کمتر.**

چه مشکل اصلی توسط OCSP حل می‌شه:

فهمیدیم OCSP دقیقاً برای حل مشکل **سنگینی، کندی و عدم‌کارایی CRL در شرایط Real-Time** به وجود اومده. وقتی گواهی‌ای فوراً لغو می‌شه (مثلاً به خاطر دزدیده شدن private key)، باید خیلی سریع این موضوع به مرورگرها اطلاع داده بشه. در اینجا OCSP با پاسخ‌دهی سریع و هدفمند، مشکل تأخیر و سنگینی CRL رو حل می‌کنه.

1.2.2 OCSP Stapling & OCSP Must-Staple

سوال : OCSP ساده (Online Certificate Status Protocol) چه کاری می‌کند؟

در واقع OCSP یک روش آنلاین برای بررسی اعتبار گواهی‌های دیجیتال است. وقتی مرورگر یا کلاینت به یک سایت متصل می‌شود، باید مطمئن شود که گواهی دیجیتال سایت معتبر و لغو نشده است. در OCSP معمولی، مرورگر مستقیماً به سرور OCSP (که معمولاً متعلق به مرجع صادرکننده گواهی یا CA است) درخواست می‌فرستد و وضعیت گواهی را بررسی می‌کند.

مشکلات و چالش‌های OCSP ساده:

1. **تاخیر در اتصال:** چون مرورگر باید قبل از برقراری اتصال ایمن، منتظر پاسخ OCSP بماند، زمان لود سایت افزایش می‌یابد.
2. **نقض حریم خصوصی:** چون مرورگر مستقیماً با سرور OCSP تماس می‌گیرد، آن سرور متوجه می‌شود که کاربر به کدام وب‌سایت مراجعه کرده، که این موضوع برای حفظ حریم خصوصی مناسب نیست.
3. **وابستگی به دسترس‌پذیری سرور OCSP:** اگر سرور OCSP در دسترس نباشد، مرورگرها معمولاً اتصال را ادامه می‌دهند (fail-soft)، که باعث کاهش امنیت می‌شود.
4. **مستعد حمله‌های man-in-the-middle:** در بعضی موارد، اگر ارتباط بین مرورگر و OCSP محافظت نشود، ممکن است مهاجم پاسخ‌های OCSP را دستکاری کند.

سوال : OCSP Stapling چگونه این مشکلات را حل کرد؟

در OCSP Stapling، دیگر این مرورگر نیست که مستقیماً به OCSP سرور متصل می‌شود. بلکه خود **وب‌سرور** به‌صورت دوره‌ای پاسخ OCSP معتبر را از CA دریافت می‌کند و آن را در هنگام TLS Handshake به مرورگر **ضمیمه می‌کند (Staple)**.

مزایای OCSP Stapling:

- **افزایش سرعت:** مرورگر نیازی به تماس مجدد ندارد، پس اتصال سریع‌تر برقرار می‌شود.
- **حفظ حریم خصوصی:** چون تماس مستقیم با OCSP توسط مرورگر حذف شده، اطلاعات مرور کاربر محرمانه می‌ماند.
- **مقاومت بهتر در برابر قطع OCSP سرور:** چون پاسخ قبلاً گرفته شده و کش شده، موقتی بودن قطعی مشکلی ایجاد نمی‌کند.

سوال : OCSP Must-Staple چیست و چه مزیتی دارد؟

دراصل OCSP Must-Staple یک افزونه است که در گواهی دیجیتال قرار می‌گیرد و به مرورگر اعلام می‌کند:

> "من فقط در صورتی معتبر هستم که گواهی Stapled OCSP همراه آن باشد."

این یعنی مرورگر الزاماً باید پاسخ OCSP را دریافت کند وگرنه اتصال را رد می‌کند و چیزی که رخ میدهد (fail-hard) است.

چالش OCSP Stapling و Must-Staple:

- اگر سرور درست پیکربندی نشده باشد یا نتواند پاسخ OCSP معتبر را Staple کند، اتصال قطع می‌شود.
- بعضی سرورها یا اپلیکیشن‌ها هنوز از این قابلیت پشتیبانی کامل ندارند.
- نیاز به زیرساخت مناسب در سرور برای دریافت و ذخیره منظم پاسخ‌های OCSP.

مزایای ترکیبی OCSP Stapling و Must-Staple نسبت به OCSP ساده:

ویژگی	OCSP ساده	OCSP Stapling	Must-Staple + Stapling
سرعت	کم	زیاد	زیاد
حریم خصوصی	ضعیف	خوب	عالی
امنیت در برابر قطع	پایین (fail-soft)	خوب	بالا (fail-hard)
نیاز به مرورگر فعال	بله	خیر	خیر
کنترل از سمت سرور	ندارد	دارد	اجباری

1.2.3 Trust in Root CAs

در ساختار PKI (Public Key Infrastructure)، پایه‌ی اصلی اعتماد روی اینترنت، **مراکز صدور گواهی (Certificate Authorities یا CAs)** هستند. اما سؤال اینجاست که:

> "اگر قراره به گواهی‌هایی که توسط CAs صادر می‌شن اعتماد کنیم، خود اون CAs رو از کجا بشناسیم؟"

ریشه‌ی اعتماد (The Chain of Trust)

کل این سیستم روی مفهومی به نام **زنجیره اعتماد (Chain of Trust)** کار می‌کنه. زنجیره از اینجا شروع میشه:

1.لایه اول: **Root CA:** یک نهاد معتبر که گواهی دیجیتال صادر می‌کنه و خودش‌گواه (self-signed) هست. این یعنی خودش گواهی خودش رو امضا کرده.

2. لایه دوم: **Intermediate CAs:** معمولاً Root CA برای کاهش خطر، گواهی‌های معتبر رو از طریق CAهای میانی صادر می‌کنه.
3. لایه سوم : **سایت یا سرور:** در نهایت، گواهی SSL/TLS به یک وب‌سایت یا سرویس تعلق می‌گیره که مرورگر قراره بهش وصل شه.

چگونه مرورگر به Root CA اعتماد می‌کنه؟

جواب ساده‌اش اینه: **مرورگر از قبل اون Root CAها رو می‌شناسه.**

لیست کلیدهای عمومی Root CAs:

مرورگرهایی مثل Edge ،Firefox ،Chrome یا سیستم‌عامل‌هایی مثل Windows و macOS، از قبل یک فهرست از گواهی‌های معتبر Root CAs رو به‌صورت **داخلی (Built-in)** داخل خودشون دارن.

به این می‌گن **Trusted Root Store**.

مثلاً:

- ویندوز: Microsoft Trusted Root Store
- فایرفاکس: Mozilla CA Certificate Program

حالا یعنی چی اعتماد از قبل وجود داره؟

یعنی شرکت‌هایی مثل Google ،Mozilla یا Microsoft خودشون تصمیم می‌گیرن کدوم CA قابل اعتماده و کلید عمومی اون رو تو لیست‌شون قرار می‌دن.

برای اینکه یه CA وارد این لیست بشه باید:

- مراحل دقیق امنیتی و ممیزی (audit) رو بگذرونه.
- سیاست‌های صدور گواهی شفافی داشته باشه.
- مورد تایید یک یا چند نهاد ممیزی (مثلاً WebTrust) قرار بگیره.

وقتی مرورگر به یک سایت متصل میشه چه اتفاقی می‌افته؟

1. سایت، گواهی خودش رو به مرورگر می‌فرسته.
2. مرورگر بررسی می‌کنه آیا این گواهی توسط یک Intermediate CA امضا شده؟
3. بعد می‌ره بالا تا برسه به Root CA.
4. اگر کل این زنجیره معتبر بود و Root CA هم در Trusted Store وجود داشت ← اتصال امن برقرار می‌شه.

در کل مرورگرها **به‌طور پیش‌فرض** به Root CAهایی اعتماد دارن که کلید عمومی اون‌ها در لیست داخلی (Trusted Root Store) ثبت شده. هویت CAها هم از طریق بررسی‌های امنیتی، auditها، و سیاست‌های شفاف‌شون تأیید می‌شه.

در واقع، **اعتماد در اینترنت یک‌بار برای همیشه از طریق سازندگان سیستم‌عامل و مرورگرها شکل گرفته، و کاربران نهایی بدون اینکه متوجه بشن، دارن از این ساختار استفاده می‌کنن.**

1.3.1 POODLE attack

حمله POODLE چیه و چطوری کار می‌کنه؟

درواقع POODLE یک حمله‌ی **cryptographic padding oracle** هست که در سال ۲۰۱۴ توسط تیم امنیتی گوگل کشف شد. این حمله به طور خاص، به **SSL 3.0** هدف می‌گیره.

مکانیزم حمله این‌جوریه:

-خود SSL 3.0 از یک نوع padding استفاده می‌کنه که **ساختار مشخصی نداره**.

- مهاجم با استفاده از یک **man-in-the-middle (MITM)** می‌تونه داده‌های رمزنگاری‌شده رو دستکاری کنه.
- با ارسال مکرر پیام‌های دستکاری‌شده و مشاهده‌ی پاسخ‌ها، می‌تونه به‌تدریج محتوای اصلی پیام (مثل کوکی session) رو **بایت‌به‌بایت حدس بزنه**.

اصلا POODLE به کجای فرآیند TLS حمله می‌کنه؟

این حمله به **لایه Record Protocol** در TLS/SSL هدف می‌زنه، جایی که داده‌ها رمزنگاری می‌شن و padding برای کامل کردن بلاک‌های رمز استفاده می‌شه.

و SSL 3.0 در بررسی padding ضعیف عمل می‌کنه، و اطلاعات زیادی از طریق پاسخ‌ها به مهاجم لو می‌ره (یعنی یه oracle درست می‌شه).

اثرات احتمالی حمله POODLE

- مهاجم می‌تونه **کوکی‌های session کاربر** رو بدزده.
- امکان **شبیه‌سازی نشست‌های معتبر** (session hijacking) وجود داره.
- برای اجرا شدن این حمله، نیاز به اینه که **مرورگر و سرور ارتباط رو به SSL 3.0 داون‌گرید کنن**.

چطوری جلوش گرفته شد؟

1. غیرفعال کردن کامل SSL 3.0

شرکت‌هایی مثل Mozilla ،Google، و Microsoft اعلام کردن که باید SSL 3.0 رو به طور کامل **غیرفعال کرد**، چون ذاتاً ناامن شده.

2. استفاده از TLS 1.2 یا بالاتر

نسخه‌های جدید TLS مثل TLS 1.2 و TLS 1.3 به جای block cipher با padding، از الگوریتم‌های امن‌تر مثل **AEAD (مثل GCM یا ChaCha20-Poly1305)** استفاده می‌کنن.

3. افزودن گزینه‌های امنیتی به مرورگرها و سرورها

امکاناتی مثل `TLS_FALLBACK_SCSV` معرفی شد تا از داون‌گرید شدن پروتکل جلوگیری بشه.

1.3.2 CA Design

۱. ساختار اصلی سیستم

استفاده از CA داخلی (Internal Certificate Authority)

در شبکه‌های سازمانی، استفاده از یک **CA داخلی** برای صدور گواهی‌ها بسیار رایج است. ما از **Active Directory Certificate Services (AD CS)** استفاده می‌کنیم، چون:

- با Windows Server ادغام می‌شود.
- به صورت خودکار گواهی برای کاربران و دستگاه‌ها صادر می‌کند.
- از Group Policy برای انتشار خودکار گواهی‌ها پشتیبانی می‌کند.

ساختار PKI پیشنهادی:

```
                    Root CA (Offline)
                          |
                    Intermediate CA
                          |
    +---------------------+---------------------+
    |                     |                     |
Web Server 1          App Server 1         DB Server 1
   ...                   ...                  ...
```

> **نکته امنیتی**: Root CA باید آفلاین نگه‌داری شود و تنها Intermediate CA گواهی صادر کند.

---

۲. سرعت و در دسترس بودن (Availability)

- برای **High Availability CA**: در Intermediate CA، از چندین سرور به صورت Load Balanced استفاده می‌کنیم تا در صورت خرابی یکی، دیگری پاسخ دهد.
- برای **Auto Enrollment**: با AD CS، کاربران و سرورها می‌توانند به صورت خودکار گواهی دریافت یا تمدید کنند.
- برای **Monitoring Tools**: از ابزارهایی مانند Microsoft System Center یا Zabbix برای مانیتورینگ وضعیت CA و OCSP استفاده می‌کنیم.

---

۳. بررسی اعتبار و ابطال گواهی‌ها

استفاده از OCSP Stapling + OCSP Must-Staple:

- استفاده از **OCSP Stapling** کمک می‌کند که سرور، وضعیت اعتبار گواهی را با سرعت بالا ضمیمه (Staple) کند و به مرورگر ارائه دهد.
- استفاده از **OCSP Must-Staple** تضمین می‌کند که اگر وضعیت ضمیمه نباشد، مرورگر اتصال را قطع کند.

> به‌جای لیست‌های حجیم CRL، استفاده از OCSP باعث **افزایش سرعت و کاهش پهنای باند مصرفی** می‌شود.

آرشیو و ثبت گواهی‌ها

- ثبت تمام گواهی‌های صادرشده در یک **LDAP Directory داخلی**، برای بررسی‌ها و پیگیری بعدی استفاده می‌شود.
- این ساختار امکان جست‌وجوی سریع گواهی‌ها و وضعیتشان را فراهم می‌کند.

Blockchain/BitCoin.2

2.1 Current Bitcoin Mining Difficulty

در تاریخ 15 خرداد 1404، سختی استخراج بیت‌کوین (Bitcoin Mining Difficulty) به **۱۲۶.۹۸ تریلیون** رسیده است.

سختی استخراج چیست؟

سختی استخراج معیاری است که نشان می‌دهد یافتن یک هش معتبر برای بلاک جدید چقدر دشوار است. هرچه این عدد بالاتر باشد، ماینرها باید تلاش بیشتری برای یافتن هش مناسب انجام دهند.

ارتباط سختی با تعداد صفرهای ابتدایی در هش هدف

در فرآیند استخراج، ماینرها باید هشی پیدا کنند که مقدار آن کمتر از مقدار هدف (target) باشد. با افزایش سختی، مقدار هدف کاهش می‌یابد، که به معنای افزایش تعداد صفرهای ابتدایی در نمایش هگزادسیمال هش است.

در حال حاضر، با سختی فعلی، هش معتبر باید دارای **۱۹ یا بیشتر صفرهای ابتدایی** در نمایش هگزادسیمال خود باشد.

2.2.1

در حال حاضر، (Mining Pools) بخش عمده‌ای از توان محاسباتی شبکه بیت‌کوین را در اختیار دارند. بر اساس داده‌های موجود، توزیع توان شبکه بین استخرهای مختلف به شرح زیر است:

توزیع توان شبکه بین استخرهای استخراج بیت‌کوین

رتبه	نام استخر	سهم از توان شبکه
1	Foundry USA	29.7%
2	AntPool	16.5%
3	F2Pool	10.4%
4	SpiderPool	5.7%
5	MARA Pool	5.4%
6	Luxor	3.5%
7	سایر استخرها	17.2%

در مجموع، (Mining Pools) حدود **82.8**٪ از کل توان محاسباتی شبکه بیت‌کوین را کنترل می‌کنند. این می‌تواند نگرانی‌هایی درباره تمرکزگرایی و امنیت شبکه ایجاد کند، زیرا در صورت تسلط یک یا چند استخر بر بیش از 50٪ توان شبکه، امکان حملات مانند "حمله 51٪" وجود دارد.

2.2.2

به نظر من اینکه Foundry USA نزدیک به ۳۰٪ از قدرت شبکه بیت‌کوین رو در اختیار داره، یه زنگ خطر محسوب میشه. درستِ که هنوز به ۵۱٪ نرسیده، ولی همین که یک نهاد انقدر قدرت داره، با اصل غیرمتمرکز بودن بیت‌کوین یه جورایی تناقض داره.

بیت‌کوین قرار بود یه سیستم باز و غیرمتمرکز باشه که هیچ‌کس نتونه روش کنترل کامل داشته باشه. ولی حالا داریم می‌بینیم که یکی‌دو تا استخر دارن تبدیل می‌شن به "غول‌های قدرت"، و این یه جورایی شبیه متمرکز شدنِ دوبارەست.

اگه فرض کنیم یه روزی چندتا از این استخرها تصمیم بگیرن با هم همکاری کنن، اون وقت احتمال حمله ۵۱٪ واقعاً جدی میشه. اینطوری امنیت شبکه آسیب می‌بینه، مخصوصاً برای تراکنش‌های بزرگ یا شبکه‌های لایه دوم مثل Lightning.

پس بله، من فکر می‌کنم هرچند الان مستقیم خطر حمله وجود نداره، ولی این سطح از تمرکز می‌تونه در آینده به امنیت و حتی اعتبار شبکه بیت‌کوین آسیب بزنه.

2.2.3

در هفتهٔ جاری (اوایل ژوئن ۲۰۲۵)، اندازهٔ میانگین هر بلاک بیت‌کوین حدود **۱.۴۰ مگابایت** بوده است.

این اندازه نشان‌دهندهٔ میزان تراکنش‌ها و فعالیت‌های شبکه در این بازهٔ زمانی است. با توجه به محدودیت‌های پروتکل بیت‌کوین و استفاده از فناوری‌هایی مانند SegWit، اندازهٔ بلاک‌ها می‌تواند تا حدود ۴ مگابایت افزایش یابد، اما میانگین فعلی نشان می‌دهد که بلاک‌ها به‌طور کامل پر نمی‌شوند.

Kerberos .3

3.1.1

۱. پشتیبانی از ticket و قابلیت‌های مرتبط

Kerberos V4:
در نسخه چهارم، تنها پشتیبانی پایه از بلیط (Ticket) وجود دارد و قابلیت‌هایی نظیر تمدید، ارسال مجدد (Forwarding) یا اعمال زمان‌بندی‌های خاص بر بلیط‌ها در نظر گرفته نشده است. این باعث می‌شود کنترل بر اعتبارسنجی کاربر محدود باشد.

Kerberos V5:
در نسخه پنجم، ساختار بلیط‌ها پیشرفته‌تر شده و امکان renew، forward و حتی postdate کردن بلیط‌ها فراهم شده است. این ویژگی‌ها انعطاف‌پذیری بیشتری در مدیریت نشست‌ها و احراز هویت‌های طولانی‌مدت یا چندگانه فراهم می‌کند، که برای محیط‌های بزرگ و توزیع‌شده حیاتی است.

---

۲. پشتیبانی از احراز هویت بین دامنه‌ای (Cross-Realm)

Kerberos V4:
نسخه ۴ فقط از احراز هویت ساده بین دو حوزه (Realm) پشتیبانی می‌کند و امکان انتقال اعتماد به‌صورت transitive وجود ندارد. یعنی اگر Realm A به Realm B اعتماد داشته باشد و B به C، لزوماً A نمی‌تواند به C اعتماد کند.

Kerberos V5:
در نسخه پنجم، احراز هویت بین حوزه‌ای به صورت transitive پیاده‌سازی شده است. این یعنی در مثال بالا، A می‌تواند به C اعتماد کند، به شرط وجود یک زنجیره‌ی قابل‌اطمینان از اعتماد بین آن‌ها. این قابلیت برای محیط‌های سازمانی و شبکه‌های بزرگ ضروری است.

---

۳. رمزنگاری و انعطاف‌پذیری الگوریتم‌ها

Kerberos V4:
رمزنگاری در نسخه چهارم تنها مبتنی بر الگوریتم **DES (Data Encryption Standard)** است. این الگوریتم امروزه ناامن تلقی می‌شود و نمی‌تواند نیازهای امنیتی مدرن را برآورده کند.

Kerberos V5:
نسخه پنجم از ساختار قابل‌توسعه‌ای استفاده می‌کند که امکان استفاده از هر الگوریتم رمزنگاری دلخواه)مانند AES یا (RC4 را فراهم می‌سازد. در این نسخه، متن رمزشده شامل یک شناسه رمزنگاری (encryption identifier) است که مشخص می‌کند از چه الگوریتمی استفاده شده. این طراحی باعث افزایش سازگاری با استانداردهای جدید و انعطاف‌پذیری امنیتی بیشتر می‌شود.

3.1.2

استفاده از PKINIT در Kerberos V5

بررسی از دیدگاه فنی، امنیتی و عملیاتی

مشکل در نسخه‌های قبلی (Kerberos V4 و حتی V5 بدون PKINIT)

در Kerberos نسخه‌های اولیه، از **کلید متقارن (symmetric key)** برای احراز هویت اولیه کاربر به KDC (Key Distribution Center) استفاده می‌شود. این روش با چالش‌های زیر مواجه بود:

1. **نیاز به توزیع امن کلیدهای پیش‌اشتراکی:**
   هر کاربر باید یک کلید مشترک با KDC داشته باشد که معمولاً از رمز عبور استخراج می‌شود. این کلید باید به‌صورت امن توزیع شود که در مقیاس‌های بزرگ (مثلاً سازمانی یا بین‌سازمانی) بسیار دشوار و ناکارآمد است.
2. **ضعف در امنیت رمز عبور:**
   اگر رمز عبور ضعیف باشد، حملات brute-force یا dictionary می‌توانند باعث افشای کلید رمزنگاری و در نتیجه جعل بلیط‌ها شوند.

هدف از معرفی PKINIT در Kerberos V5

در**PKINIT** به Kerberos این امکان را می‌دهد که از **رمزنگاری کلید عمومی (Public Key Cryptography)** برای مرحله‌ی اولیه‌ی احراز هویت استفاده کند. در این روش، به‌جای کلید متقارن از **گواهی‌نامه دیجیتال (Digital Certificate)** و جفت کلید عمومی/خصوصی استفاده می‌شود.

مزایای PKINIT از دیدگاه عملیاتی (Practical):

1. **حذف نیاز به کلیدهای پیش‌اشتراکی برای هر کاربر:**
   دیگر نیازی نیست که هر کاربر یک کلید مشترک با KDC داشته باشد. کافیست KDC به CA (مرجع صدور گواهی) اعتماد داشته باشد.
2. **افزایش مقیاس‌پذیری:**
   در محیط‌های بزرگ (مانند شبکه‌های سازمانی با هزاران کاربر یا ساختارهای cross-realm) مدیریت کلیدهای متقارن مشکل‌ساز است. با PKINIT، تنها صدور و مدیریت گواهی‌ها کافیست.
3. **سازگاری با زیرساخت‌های PKI (Public Key Infrastructure):**
   می‌توان از گواهی‌نامه‌های X.509 استاندارد استفاده کرد، که در بسیاری از سازمان‌ها از قبل موجود هستند.

مزایای PKINIT از دیدگاه امنیتی (Security):

1. **احراز هویت قوی‌تر و امن‌تر:**
   در PKINIT، حتی اگر رمز عبور کاربر فاش شود، بدون دسترسی به کلید خصوصی او نمی‌توان مرحله‌ی احراز هویت را کامل کرد.
2. **مقاومت در برابر حملات brute-force:**
   به‌دلیل عدم استفاده مستقیم از رمز عبور در رمزنگاری، حملات مبتنی بر حدس زدن رمز عبور بی‌اثر می‌شوند.
3. **پشتیبانی از امضای دیجیتال و صحت پیام:**
   با استفاده از کلید خصوصی، کاربر می‌تواند پیام‌ها را امضا کند و KDC با کلید عمومی صحت آن را بررسی می‌کند. این قابلیت یک لایه امنیتی مهم در برابر جعل پیام یا replay attacks ایجاد می‌کند.

3.2.1

```
$K_{KDC}$ = master key
$K_a$ = $h$(Alice password)
$S_A$ = session key
TGT = E("Alice", $S_A$, $K_{KDC}$)
authenticator1 = E(timestamp, $S_A$)
REQUEST= (TGT, authenticator)
TicketToBob = E(“Alice”, $K_{AB}$, $K_B$)
REPLY=E(“Bob”, $K_{AB}$, ticket to Bob, $S_A$)
authenticator2 = E(timestamp, $K_{AB}$)
| sender | message  | receiver  |receiver action|
|:-:|:-:|:-:|:-:|
|Alice| "Alice" wants a TGT | KDC (AS) | creates $S_A$ and TGT
|KDC (AS)| E($S_A$, TGT, $K_A$)|Alice|decrypt it using  $K_A$, get the $S_A$ and creates authenticator1
|Alice|I want to talk to bob,REQUEST|KDC (TGS)| decrypts the TGT, gets the $S_A$ and verifies the authenticator1
then creates TicketToBob and $K_{AB}$ and puts them in REPLY
|KDC (TGS)| REPLY | Alice| gets the TicketToBob and $K_{AB}$
|Alice| TicketToBob, authenticator2 | Bob| decrypts the TicketToBob gets the $K_{AB}$ then verifies the
authenticator2
|Bob| E(timestamp+1, $K_{AB}$)|Alice| verifies the timestamp
```

3.2.2

TGT:

با کلید K_{KDC} رمز می‌شود تا بعدا Alice نتواند خود را جای فرد دیگری جا بزند. اگر با کلید دیگری باشد، مثلا S_A یا K_A در این‌صورت Alice می‌تواند نام خود را تغییر دهد مثلا به Bob و به جای آن اهراز هویت شود. در واقعیت TGT باید ضمانت کند فقط Alice بتواند با آن اهراز هویت شود و اهزار هویت فقط باید توسط KDC انجام شود و نه کس دیگری. رمزنگاری با K_{KDC} می‌تواند این را ممکن بسازد.

TicketToBob:

طراحی Kerberos به گونه ای است که Stateless باشد و نیز نیاز نباشد مستقیما با سرویس ها ارتباط برقرار کند. همچنین سرویس ها نیازی نداشته باشند تا با KDC ارتباط برقرار کنند. در این حالت سرویس ها با یک بررسی متوجه اهراز هویت Alice می‌شوند. همچنین این Alice است که شروع کننده ارتباط است و هر وقت که میخواهد باید با یک authenticator2 خودش را اثبات کند. اگر TicketToBob به Bob ارسال می‌شد، آنگاه سرویس باید منتظر Alice می‌ماند. این کار به پیاده سازی Stateless و راحت تر کمک می‌کند.

3.3.1

حمله‌ی Golden Ticket از طریق جعل تیکت سرویس (Ticket Granting Ticket - TGT) در پروتکل Kerberos انجام می‌گیرد. مهاجم با داشتن کلید اصلی (Master key) سرویس Kerberos (کلید رمزنگاری مربوط به حساب krbtgt در Active Directory)، می‌تواند یک TGT جعلی تولید کند که مورد قبول کنترل‌کننده‌ی دامنه (Domain Controller) باشد. مراحل اصلی این حمله به شرح زیر است:

1. **دستیابی به کلید حساب krbtgt**: این کلید معمولاً با دسترسی به حافظه (از طریق ابزارهایی مثل Mimikatz) از کنترل‌کننده‌ی دامنه استخراج می‌شود.
2. **ساخت تیکت جعلی (TGT)**: با استفاده از کلید krbtgt، مهاجم می‌تواند تیکت‌های TGT دلخواهی برای هر کاربری (حتی کاربران با دسترسی بالا مثل domain admin) ایجاد کند.
3. **استفاده از تیکت جعلی برای دسترسی به سرویس‌ها**: مهاجم با این تیکت جعلی می‌تواند به منابع مختلف در شبکه بدون محدودیت زمانی یا سطح دسترسی وارد شود.

اطلاعات مورد نیاز:

- کلید رمزنگاری krbtgt
- نام دامنه و SID دامنه
- نام کاربری هدف (در صورت ساخت تیکت برای کاربر خاص)

3.3.2

هدف اصلی این حمله، **TGT و سرویس صدور تیکت (Ticket Granting Service - TGS)** در Kerberos است. بخش آسیب‌پذیر، حساب **krbtgt** است که مسئول رمزنگاری و تأیید اعتبار TGT است.

چرا آسیب‌پذیر محسوب می‌شود؟

- کلید krbtgt برای تمام TGTها مشترک است؛ در صورت افشای این کلید، امکان جعل تیکت برای هر کاربری وجود دارد.
- کنترل‌کننده دامنه (DC) اعتبار این تیکت جعلی را بررسی نمی‌کند چون به کلید krbtgt اعتماد کامل دارد.

3.3.3

راهکار ۱: چرخش دوره‌ای کلید krbtgt

- **مزایا**: باعث باطل شدن تمام تیکت‌های جعلی تولیدشده‌ی پیشین می‌شود.
- **معایب**: نیاز به هماهنگی دقیق در محیط AD دارد. اجرای نادرست ممکن است باعث اختلال در اعتبارسنجی شود.

راهکار ۲: نظارت پیشرفته بر ورودها و فعالیت‌ها با SIEM

- **مزایا**: امکان شناسایی فعالیت‌های مشکوک با تحلیل الگوهای رفتاری.
- **معایب**: نیاز به منابع، تخصص و پیاده‌سازی سامانه‌های نظارتی پیشرفته دارد.

3.4

الف)

سیستم Kerberos برای جلوگیری از حملات تکراری (Replay Attacks) به شدت به همزمانی دقیق ساعت بین کلاینت و سرور متکی است. اگر زمان سیستم‌ها هماهنگ نباشد، تیکت صادر شده ممکن است به عنوان نامعتبر شناسایی شود یا امکان جعل تیکت ساده‌تر گردد. این همزمانی معمولاً از طریق پروتکل NTP تأمین می‌شود.

ب)

- **klist**:

لیست تیکت‌های Kerberos موجود در کش کاربر را نمایش می‌دهد، شامل:

- تیکت های TGT فعال
- تیکت‌های سرویس
- زمان انقضا و صدور

- **kinit**:

برای گرفتن یک TGT جدید از KDC به‌کار می‌رود. با اجرای آن و وارد کردن رمز عبور، سیستم یک تیکت جدید در کش قرار می‌دهد.
