

# TLS/SSL.1

## 1.1.1 TLS/SSL Handshake Process: Key Exchange and Authentication

در ارتباط امن بین مرورگر و سرور که از SSL یا TLS استفاده می‌کنند، اولین کاری که انجام می‌شود، تبادل اطلاعات برای ساختن کلید رمزگاری بین دو طرفه. این مرحله که بهش می‌گذرد، چند تا پیام اصلی دارد که هر کدام یه نقش مهم دارند:

### 1. ClientHello:

اول از همه، کلاینت (مثلاً مرورگر کاربر) یه پیام می‌فرسته به اسم ClientHello. توی این پیام اطلاعاتی مثل نسخه TLS که پشتیبانی می‌کنند، لیست الگوریتم‌های رمزگاری که بله، یه عدد تصادفی (که بعداً برای ساختن کلید استفاده می‌شود) و چند تا اطلاعات دیگه فرستاده می‌شود. اینطوری کلاینت نشون می‌ده که آماده شروع ارتباط امن هست.

### 2. ServerHello:

بعد سرور جواب می‌ده با ServerHello. توی این پیام، سرور یکی از الگوریتم‌های پیشنهادی کلاینت را انتخاب می‌کند و یه عدد تصادفی دیگه هم بله اضافه می‌کند. این عدد هم کنار عدد کلاینت استفاده می‌شود برای ساختن کلید نهایی.

### 3. Certificate:

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

### 4. ServerKeyExchange (اگه لازم باشه):

اگه سرور از الگوریتم‌های خاصی مثل Diffie-Hellman استفاده کند، یه پیام جداآگونه هم می‌فرسته که پارامترهای لازم برای این الگوریتم رو دارد. اینجوری دو طرف می‌توانند یه کلید مشترک بسازن بدون اینکه کسی بتوانه وسط کار شنود کند.

### 5. ClientKeyExchange:

اینجا کلاینت یه اطلاعات مهم به اسم pre-master secret تولید می‌کند و می‌فرسته برای سرور. این اطلاعات معمولاً با کلید عمومی سرور رمز می‌شنند که فقط خود سرور می‌توانه بازش کند. با کمک این pre-master secret و اون دو تا عدد تصادفی قبلی، کلاینت و سرور کلید اصلی رو می‌سازن.

### 6. ChangeCipherSpec + Finished:

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

در کل، این فرآیند باعث می‌شود دو طرف بدون اینکه کسی دیگه بتوانه شنود کند، یه کلید رمز مشترک داشته باشند که بعدش از اون برای رمز کردن اطلاعات استفاده می‌کنند.

## 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 رمزگاری مثل AES رمز می‌شود، بعده یه MAC یا همون کد احراز صحت (مثلاً با HMAC-SHA256) برآش تولید می‌شود و به آخر پیام اضافه می‌شود.

ولی من می‌خواهم یه روش ساده‌تر پیشنهاد بدم که هم محرمانگی رو تأمین کنند 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 (مثالاً یه کلید ۲۵۶ بیتی که از راه امن منتقل شده).
۲. وقتی می‌خوایم یه داده (مثالاً یک پیام متنتی) رو ارسال کنیم:
  - اول، اون داده رو با استفاده از یه الگوریتم مقارن (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 می‌توانه 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 بخش:

- 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. بعده، 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 قدمی استفاده می‌شد، ولی در TLS 1.2 امکان استفاده از الگوریتم‌های جدیدتر (مثل AES-GCM و SHA-256) اضافه شد.
- توی TLS 1.2، ساختار MAC و encryption بهصورت کامل قابل تنظیم شدن. یعنی می‌شه الگوریتم‌های بهروزتری رو انتخاب کرد که هم امن‌تر، هم سریع‌تر.
- همچین 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 مزیت‌های 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 مقارن، MAC و کلید عمومی استفاده می‌کرد.	آسیب‌پذیر به حمله POODLE. امنیت نمی‌شود.

نسخه	ویژگی‌ها	مشکلات یا ضعف‌ها
SAL (1999)	SSL 3.0 با الگوریتم‌های رمزنگاری بهتر (مثل HMAC) ساختار handshake امن‌تر شد.	در برابر حملات مثل BEAST آسیب‌پذیر.
SAL (2006)	بهبود در مدیریت Initialization Vector (IV) برای جلوگیری از حمله BEAST.	هنوز از الگوریتم‌های رمزنگاری قدیمی استفاده می‌کرد. امنیت کافی نداشت.
SAL (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.
SAL (2018)	حذف الگوریتم‌های نامن مثل RC4، حذف RSA Key Exchange، کاهش تعداد round-trip در handshake. فقط الگوریتم‌های قوی باقی موندند.	امن‌ترین و سریع‌ترین نسخه تا به امروز.

## بهبودهای مهم در TLS 1.3 نسبت به نسخه‌های قبلی

### 1. حذف الگوریتم‌های ضعیف و نامن:

- الگوریتم‌هایی مثل SHA-1، RC4، MD5، و حتی RSA برای key exchange حذف شدند.
- این کار باعث شد حملات کلاسیکی مثل MITM و downgrade key recovery بی‌اثر بشن.

### 2. ساده‌سازی handshake:

- در TLS 1.2، راهاندازی ارتباط معمولًا دو round-trip لازم داشت، ولی در 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 به صورت اجباری:

- فقط الگوریتم‌های تبادل کلید به صورت Ephemeral Diffie-Hellman (مثل موقتی) مجاز هستند.
- این یعنی اگه حتی کلید خصوصی لو برده، ارتباطات گذشته همچنان امن می‌مونند.

### 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	TLS 1.3 در
سرعت اتصال	کنترل (handshake 2 مرحله)	سریع‌تر (handshake 1 مرحله)
امنیت	امکان حملات downgrade و استفاده از الگوریتم ضعیف	حذف کامل الگوریتم‌های نامن
پشتیبانی از Forward Secrecy	اختیاری بود	اجباری و پیش‌فرض
حجم کد اجرایی	بیچیده و بزرگ	садه‌تر 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 وب‌سورورها 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 صادرکننده 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 بیننه آیا گواهی مورد نظر توی اون لیست هست یا نه.

### مشکل CRL چیه؟

- سنگینی فایل: اگه تعداد زیادی گواهی لغو شده باشه، لیست حجم 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 مهمنشه.
- بار زیاد روی سرور CA: همهی کلاینت‌ها باید این فایل بزرگ رو بگیرن، که فشار زیادی به سرور می‌باره.

## سوال OCSP (Online Certificate Status Protocol) چیست؟

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

:CRL نسبت به OCSP مزیت

- 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 عدم کارایی CRL در شرایط Real-Time با وجود امده. وقتی گواهی ای 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 : ساده OCSP چه کاری میکند؟ (Online Certificate Status Protocol)

در واقع OCSP یک روش آنلاین برای اعتبار گواهی های دیجیتال است. وقتی مرورگر یا کلینت 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 چالش های OCSP ساده:

- تأخیر در اتصال: چون مرورگر باید قبل از برقراری اتصال این، منظر پاسخ OCSP بماند، زمان لود سایت افزایش میباید.
- 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.
- وابستگی به دسترس پذیری سرور OCSP: اگر سرور OCSP در دسترس نباشد، مرورگرها معمولاً اتصال را ادامه میدهند (fail-soft)، که باعث کاهش امنیت میشود.
- مستعد حمله های man-in-the-middle: در بعضی موارد، اگر ارتباط بین مرورگر و OCSP محافظت نشود، ممکن است مهاجم پاسخ های OCSP را دستکاری کند.

### سوال : چگونه این مشکلات را حل کرد؟

در OCSP Stapling، دیگر این مرورگر نیست که مستقیماً به OCSP سرور متصل میشود. بلکه خود و بسرور به صورت دوره ای 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 محروم نمیماند.
- مقاومت بهتر در برابر قطع OCSP سرور: چون پاسخ قبلاً گرفته شده و کش شده، موقتی بودن قطعی مشکلی ایجاد نمیکند.

### سوال : OCSP چیست و چه مزیتی دارد؟

در اصل 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 و گرنم اتصال را رد میکند و چیزی که رخ میدهد (fail-hard) است.

### چالش Must-Staple و OCSP Stapling

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

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

### 1.2.3 Trust in Root CAs

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

"اگر قراره به گواهی‌هایی که توسط CAs صادر می‌شون اعتماد کنیم، خود اون CAs را 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 خودش گواه (self-signed) هست. این یعنی خودش گواهی خودش رو امضا کرده.
2. لایه دوم: **Intermediate CAs**: معمولاً Root CA برای کاهش خطر، گواهی‌های معتبر را 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 Chrome، Firefox، Edge، Windows و macOS، از قبیل یک فهرست از گواهی‌های معتبر Root CAs را به صورت داخلی (Built-in) یا سیستم‌عامل‌هایی مثل Microsoft Trusted Root Store را در خودشون دارن.

به این می‌گن Trusted Root Store.

مثلًا:

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

## حالا یعنی چی اعتماد از قبل وجود داره؟

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

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 میرمه بالا تا بررسه 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 به طور پیشفرض به Root CA هایی اعتماد دارن که کلید عمومی اونها در لیست داخلی (Trusted Root Store) ثبت شده. هویت CAها هم از طریق بررسی های امنیتی، audit ها، و سیاست های شفاف شون تأیید میشه.

در واقع، اعتماد در اینترنت یکبار برای همیشه از طریق سازندگان سیستم عامل 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؟

در واقع یک حمله‌ی 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.

mekanizm حمله این جوړیه:

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

- مهاجم با استفاده از یک man-in-the-middle (MITM) میتوانه داده های رمز نگاری شده را دستکاری کنه.
- با ارسال مکرر پیام های دستکاری شده و مشاهده پاسخ ها، میتوانه به تدریج محتوای اصلی پیام (مثل کوکی session) را بازبینی کند.

#### اصلًا POODLE به کجا فرایند TLS حمله میکنه؟

این حمله به لایه Record Protocol در TLS/SSL هدف میزنه، جایی که داده ها رمز نگاری میشن و padding برای کامل کردن بلاک های رمز استفاده میشه.

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

#### اثرات احتمالی حمله POODLE

- مهاجم میتوانه کوکی های session کاربر را بدزد.
- امکان شبیه سازی نشست های معتبر (session hijacking) وجود داره.
- برای اجرا شدن این حمله، نیاز به اینه که مرورگر و سرور ارتباط رو به SSL 3.0 داون گرید کنن.

#### چطوری جلوش گرفته شد؟

##### 1. غیرفعال کردن کامل SSL 3.0

شرکت هایی مثل Microsoft، Mozilla، Google، و Google اعلام کردن که باید SSL 3.0 رو به طور کامل غیرفعال کرد، چون ذاتاً نامن شده.

##### 2. استفاده از TLS 1.2 یا بالاتر

نسخه های جدید TLS مثل TLS 1.2 و TLS 1.3 به جای padding با block cipher مثلاً GCM یا AEAD استفاده میکنن.

##### 3. افزودن گزینه های امنیتی به مرورگرها و سرورها

امکاناتی مثل TLS\_FALLBACK\_SCSV معرفی شد تا از داون گرید شدن پروتکل جلوگیری بشه.

## 1.3.2 CA Design

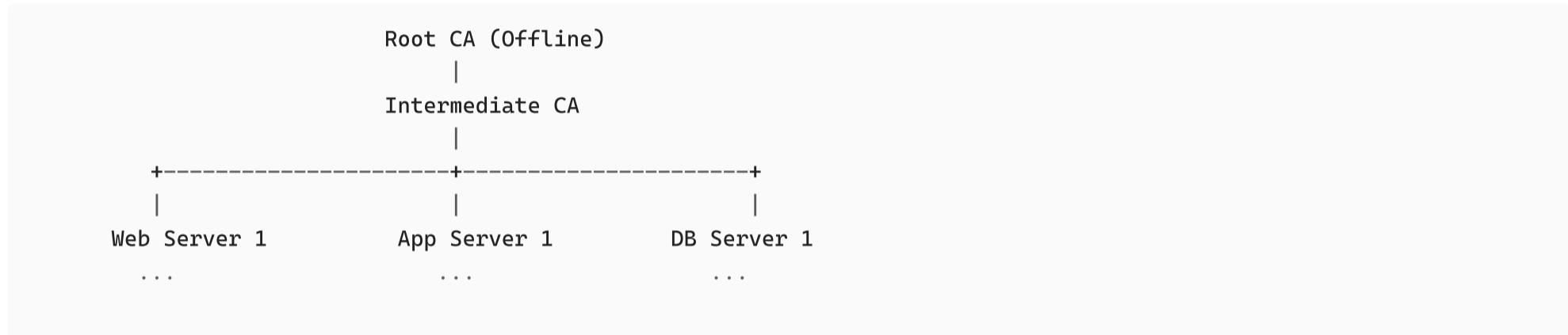
### ۱. ساختار اصلی سیستم

#### استفاده از CA داخلي (Internal Certificate Authority)

در شبکه‌های سازمانی، استفاده از یک **CA داخلي** 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:



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

### ۲. سرعت و در دسترس بودن (Availability)

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

به جای لیست‌های حجمی CRL، استفاده از OCSP باعث افزایش سرعت و کاهش پهناوری باند مصرفی می‌شود.

#### آرشیو و ثبت گواهی‌ها

- ثبت تمام گواهی‌های صادرشده در یک **LDAP Directory** داخلي، برای بررسی‌ها و پیگیری بعدی استفاده می‌شود.
- این ساختار امکان جستجوی 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 ۱۶.۹۸ 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 می‌باید، 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 استخراها مختلف به شرح زیر است:

### توزیع توان شبکه بین استخراها 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 استخراها	17.2%

در مجموع، (Mining Pools) حدود ۸۲.۸٪ 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 ۵۰٪ توان شبکه، امکان حملات مانند "حمله ۵۱٪" وجود دارد.

## 2.2.2

به نظر من اینکه Foundry USA نزدیک به ۳۰٪ از قدرت شبکه بیتکوین را در اختیار دارد، به زنگ خطر محسوب می‌شود. درست که هنوز به ۵۱٪ نرسیده، ولی همین که یک نهاد انقدر قدرت دارد، با اصل غیرتمرکز بودن بیتکوین یه جوابی تناقص دارد.

بیتکوین فرار بود یه سیستم باز و غیرتمرکز نتونه روشن کنترل کامل داشته باشه. ولی حالا داریم می‌بینیم که یکی‌دو تا استخرا دارن تبدیل می‌شن به "غول‌های قدرت"، و این یه جوابی شبیه متمرکز شدن دوباره است.

اگه فرض کنیم یه روزی چندتا از این استخراها تصمیم بگیرن با هم همکاری کنن، اون وقت احتمال حمله ۵۱٪ واقعاً جدی می‌شود. 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 هفته جاری (اوایل ژوئن ۲۰۲۵)، اندازه میانگین هر بلاک بیتکوین حدود ۴۰.۴۰ مگابایت بوده است.

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

### 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 پیشرفت‌های بیشتر 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 به صورت 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 قابل اطمینان 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 نامن‌تلقی می‌شود و نمی‌تواند نیازهای امنیتی مدرن را برآورده کند.

### Kerberos V5:

نسخه پنجم از ساختار قابل توسعه‌ای 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 (PKINIT و حتی V5 بدون Kerberos V4)

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

۱. 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 به صورت امن توزیع شود که در مقیاس‌های بزرگ (مثل سازمانی یا بین‌سازمانی) بسیار دشوار و ناکارآمد است.

۲. ضعف در امنیت رمز عبور:

اگر رمز عبور ضعیف باشد، حملات brute-force یا dictionary می‌توانند باعث افشاگری کلید رمزنگاری و در نتیجه جعل بلیط‌ها شوند.

## هدف از معرفی Kerberos V5 در PKINIT

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

## مزایای PKINIT از دیدگاه عملیاتی (Practical)

1. حذف نیاز به کلیدهای پیش اشتراکی 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 مقیاس پذیری:  
در محیطهای بزرگ (مانند شبکه های سازمانی با هزاران کاربر یا ساختارهای cross-realm) مدیریت کلیدهای متقارن مشکل ساز است. با PKINIT، تنها صدور و مدیریت گواهی ها کافیست.
3. سازگاری با زیرساخت های PKI (Public Key Infrastructure):  
می توان از گواهی نامه های X.509 استاندارد استفاده کرد، که در بسیاری از سازمان ها از قبل موجود هستند.

## مزایای PKINIT از دیدگاه امنیتی (Security)

1. احراز هویت قوی تر و امن تر:  
در PKINIT، حتی اگر رمز عبور کاربر فاش شود، بدون دسترسی به کلید خصوصی او نمی توان مرحله ای احراز هویت را کامل کرد.
2. مقاومت در برابر حملات brute-force:  
به دلیل عدم استفاده مستقیم از رمز عبور در رمز نگاری، حملات مبتنی بر حدس زدن رمز عبور بی اثر می شوند.
3. پشتیبانی از امضای دیجیتال و صحت پیام:  
با استفاده از کلید خصوصی، کاربر می تواند پیام ها را امضا کند و KDC با کلید عمومی صحت آن را بررسی می کند. این قابلیت یک لایه امنیتی مهم در برابر جعل پیام یا replay attacks ایجاد می کند.

### 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 create authenticator1
Alice	I want to talk to Bob, REQUEST	KDC (TGS)	decrypts the TGT, gets $S_A$ , verifies authenticator1, creates TicketToBob and $K_{AB}$
KDC (TGS)	REPLY	Alice	gets the TicketToBob and $K_{AB}$
Alice	TicketToBob, authenticator2	Bob	decrypts TicketToBob, gets $K_{AB}$ , verifies authenticator2
Bob	$E(timestamp+1, K_{AB})$	Alice	verifies the timestamp

### 3.2.2

#### TGT:

با کلید  $K_{KDC}$  رمز می شود تا بعدا Alice نتواند خود را جای فرد دیگری جا بزند. اگر با کلید دیگری باشد، مثلا  $S_A$  یا  $K_A$  در این صورت Alice می تواند 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 اهراز هویت فقط باید توسط KDC انجام شود و نه کس دیگری. رمز نگاری با  $K_{KDC}$  می تواند این را ممکن نسازد.

#### TicketToBob:

طرahiی Kerberos به گونه ای است که Stateless باشد و نیاز نباشد مستقیماً 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 می شوند. همچنین این Alice است که شروع کننده ارتباط است و هر وقت که می خواهد باید با یک authenticator2 خودش را اثبات کند. اگر Bob ارسال می شد، آنگاه سرویس باید منتظر Alice می ماند. این کار به پیاده سازی Stateless و راحت تر کمک می کند.

### 3.3.1

حملهی Golden Ticket از طریق جعل تیکت سرویس (Ticket Granting Ticket - TGT) در پروتکل Kerberos انجام می گیرد. مهاجم با داشتن کلید اصلی (Master key) (کلید رمزنگاری مربوط به حساب krbtgt در Active Directory)، می تواند یک TGT جعلی تولید کند که مورد قبول کنترل کننده 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 کنترل کننده دامنه استخراج می شود.
2. ساخت تیکت جعلی (TGT): با استفاده از کلید **krbtgt**، مهاجم می تواند تیکت های TGT دلخواهی برای هر کاربری (حتی کاربران با دسترسی بالا مثل domain admin) ایجاد کند.
3. استفاده از تیکت جعلی برای دسترسی به سرویس ها: مهاجم با این تیکت جعلی می تواند 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 آسیب پذیر، حساب **krbtgt** است که مسئول رمزنگاری و تأیید اعتبار TGT است.

چرا آسیب پذیر محسوب می شود؟

- کلید **krbtgt** برای تمام TGT ها مشترک است؛ در صورت افسای این کلید، امکان جعل تیکت برای هر کاربری وجود دارد.
- کنترل کننده دامنه (DC) اعتبار این تیکت جعلی را بررسی نمی کند چون به کلید **krbtgt** اعتماد کامل دارد.

### 3.3.3

راهکار ۱: چرخش دوره ای کلید **krbtgt**

- مزایا: باعث باطل شدن تمام تیکت های جعلی تولید شده می شود.
- 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 فعالیت ها با **SIEM**

- مزایا: امکان شناسایی فعالیت های مشکوک با تحلیل الگوهای رفتاری.
- معایب: نیاز به منابع، تخصص و پیاده سازی سامانه های 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 سیستم ها هماهنگ نباشد، تیکت صادر شده ممکن است به عنوان نامعتبر شناسایی شود یا امکان جعل تیکت ساده تر گردد. این همزمانی معمولاً از طریق پروتکل NTP تأمین می شود.

(ب)

- **klist:**

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

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

- **kinit:**

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

---

---