# Лекция 4

Шифрование. Распределение ключа.

# Виды шифрования

# Симметричные

- Типы
- Атаки
- Side-channel attack
- Brute force attack
- Распределение ключа

# Асимметричные

- RSA - самая массовая криптосистема с открытым ключом (асимметричная криптосистема). Повсеместно используется для подписи в составе SSL-сертификатов.

- В RSA однонаправленная функция имеет "лазейку" или "чёрный ход", знание которого позволяет легко обратить функцию, вычислить аргумент по значению. RSA основана на возведении в степень по модулю, а стойкость этой криптосистемы связана с задачей разложения числа на простые множители.

- Параметрами криптосистемы RSA являются модуль и открытая (шифрующая) экспонента - то есть, показатель степени, в которую возводится сообщение.
- Модуль в RSA является составным числом: N = pq, где p и q - достаточно большие простые числа.
- Разложение N держится в секрете. Знание этого разложения позволяет построить "лазейку", с помощью которой можно обратить однонаправленную функцию, лежащую в основе RSA.

- Секретным ключом является экспонента, обратная к шифрующей (расшифровывающая).
- В случае TLS, сервер публикует модуль N и шифрующую экспоненту e, а в секрете сохраняется соответствующая e расшифровывающая экспонента $d = e^{-1}$
- Таким образом, открытый ключ состоит из N и e, а закрытый - из (N,p,q,d).
- Открытый ключ обычно публикуется в составе сертификата TLS
- Открытый ключ служит для проверки подписей, а для их генерации необходимо знать соответствующий секретный.
- RSA позволяет зашифровать сообщение, выступая в роли асимметричного шифра.
- $C = m^e \bmod N$, где C - шифротекст, а m - открытый текст. Соответственно, зная расшифровывающую экспоненту, можно расшифровать C: $m = C^d \bmod N$

- При использовании для создания и проверки электронной подписи в TLS, RSA работает следующим образом. Сторона, генерирующая подпись, вычисляет значение хеш-функции от подписываемого сообщения (H = Hash(M)), приводит это значение к разрядности используемого модуля N и вычисляет значение подписи при помощи секретной (расшифровывающей) экспоненты: $S = H^d \bmod N$.
- Для проверки подписи требуется знать открытую часть ключа: модуль и шифрующую экспоненту e. Значение подписи возводится в степень e, а получившееся значение сравнивается с вычисленным значением хеш-функции сообщения: $S^e \bmod N = H$, если значения совпали, то подпись верна. Для вычисления корректной подписи (подделки) третьей стороне необходимо знать секретную экспоненту d, что обеспечивает защиту от подделки.

# Свойства функции хэширования

- Необратимость
- Стойкость к коллизиям

# Использования

- Хранение пароля
- Проверка эцп
- Сертификаты
- Проверка целостности, HMAC

# MAC/HMAC

- Message Authentication code – стандарт обмена данными при помощи общего ключа
- HMAC – MAC с использованием hash
- HMAC позволяет получить код аутентичности HMAC (k, text)
- Безопасность эквивалентна безопасности используемой хэш функции
- Быстрее симметричного шифрования
- Используется в TLS, IPSec

# Хэши

- В марте 2012 года вышла последняя на данный момент редакция *FIPS PUB 180-4*, в которой были добавлены функции *SHA-512/256* и *SHA-512/224*, основанные на SHA-512 (поскольку на 64-битных архитектурах SHA-512 работает быстрее, чем SHA-256)[4].
- В июле 2006 года появился стандарт RFC 4634 «Безопасные хеш-алгоритмы США (*SHA* и *HMAC-SHA*)», описывающий *SHA-1* и семейство *SHA-2*.

# Рекомендации

- Отказаться от SHA 1/ MD5
- Использовать SHA2, следить

# Использование шифрования в TLS

- TLS Record Protocol : symmetric, MAC – privacy, integrity
- TLS Handshake Protocol: communicate keys.
- Specification of key exchange is left with the upper level
- higher layers should not be overly reliant on whether TLS always negotiates the strongest possible connection between two peers. There are a number of ways in which a man-in-the-middle attacker can attempt to make two entities drop down to the least secure method they support

# Handshake Protocol – hello phase

- The client sends a ClientHello message to which the server must respond with a ServerHello message, or else a fatal error will occur and the connection will fail.
- The ClientHello and ServerHello are used to establish security enhancement capabilities between client and server. The ClientHello and ServerHello establish the following attributes: Protocol Version, Session ID, Cipher Suite, and Compression Method.
- Additionally, two random values are generated and exchanged: ClientHello.random and ServerHello.random.

# Handshake Protocol – key exchange

- the server Certificate, the ServerKeyExchange, the client Certificate, and the ClientKeyExchange.
- New key exchange methods can be created by specifying a format for these messages and by defining the use of the messages to allow the client and server to agree upon a shared secret.
- secrets that range from 46 bytes upwards

# Handshake Protocol

- клиент подключается к серверу, поддерживающему TLS, и запрашивает защищённое соединение;
- клиент предоставляет список поддерживаемых алгоритмов
- сервер выбирает из списка, предоставленного клиентом, наиболее надёжные алгоритмы среди тех, которые поддерживаются сервером, и сообщает о своём выборе клиенту;
- сервер отправляет клиенту цифровой сертификат для собственной аутентификации: имя сервера, открытый ключ, имя центра сертификации
- клиент, до начала передачи данных, проверяет валидность (аутентичность) полученного серверного сертификата, относительно имеющихся у клиента корневых сертификатов удостоверяющих центров (центров сертификации).
- для шифрования сессии используется  общий сеансовый ключ (Диффи Хеллман).

# Установление защищенного соединения

- Клиент посылает сообщение **ChangeCipherSpec**, которое указывает на то, что вся последующая информация будет зашифрована установленным в процессе подтверждения связи алгоритмом, используя общий секретный ключ.
  - Клиент посылает зашифрованной сообщение **Finished**, которое содержит хеш и MAC, сгенерированные на основе предыдущих сообщений процедуры подтверждения связи;
  - Сервер пытается расшифровать Finished-сообщение клиента и проверить хеш и MAC. Если процесс расшифровки или проверки не удаётся, подтверждение связи считается неудавшимся, и соединение должно быть оборвано;
- Сервер посылает **ChangeCipherSpec** и зашифрованное сообщение Finished, и в свою очередь клиент тоже выполняет расшифровку и проверку.
- С этого момента подтверждение связи считается завершённым, протокол установленным. Всё последующее содержимое пакетов идет с типом 23, а все данные будут зашифрованы.

# Алгоритм DH

- параметры протокола Диффи-Хеллмана (DH) должны быть подписаны.
- Если используется классический вариант, то в сообщении ServerKeyExchange передаётся значение модуля и вычисленный сервером открытый ключ DH.
- В варианте на эллиптических кривых (ECDH) - идентификатор самой кривой и, аналогично DH, открытый ключ. Параметры подписываются сервером
- В зависимости от используемой криптосистемы, подпись может быть DSA (сейчас практически не встречается), RSA или ECDSA.
- Подпись на параметрах DH очень важна, так как позволяет защитить соединение от атаки типа "Человек посередине".

# Master secret

- Основа ключей - общий секрет Master Secret - генерируется из нескольких переменных составляющих: так называемый Premaster Secret, ClientRandom и ServerRandom. Premaster Secret - согласуется в рамках обмена ключами, это либо последовательность случайных байтов, зашифрованная открытым ключом сервера, либо значение, полученное в результате обмена по протоколу Диффи-Хеллмана.
- master_secret = PRF(pre_master_secret, "master secret", ClientHello.random + ServerHello.random)[0..47];

# Методы установления ключа

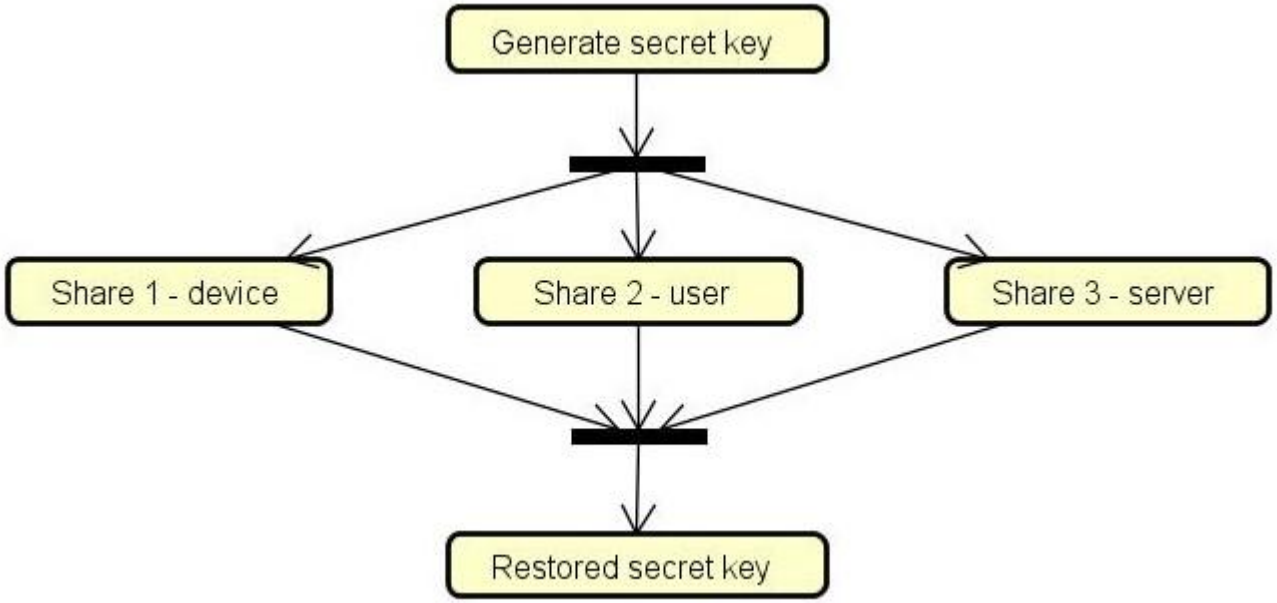
- **PSK - pre-shared key.** Схема основана на использовании общего секретного ключа и симметричной криптосистемы, при условии, что ключ был согласован заранее;
- 2. **временный ключ RSA** - его поддержка послужила основой для атаки FREAK, опубликованной в 2014 году;
- 3. **SRP** - протокол SRP (Secure Remote Password), RFC 5054. Протокол, позволяющий сгенерировать общий симметричный секретный ключ достаточной стойкости на основе известного клиенту и серверу пароля, без раскрытия этого пароля через незащищённый канал;

# Еще

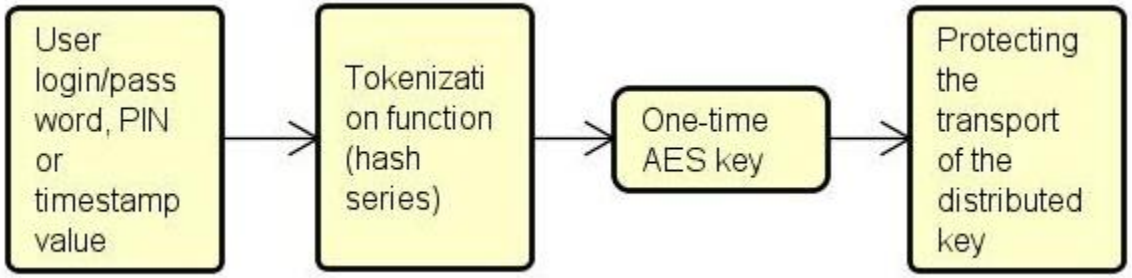
- SS
- KDE
- Trusted timestamnig
- EKE
- OTP
- ….

# Распределение ключа

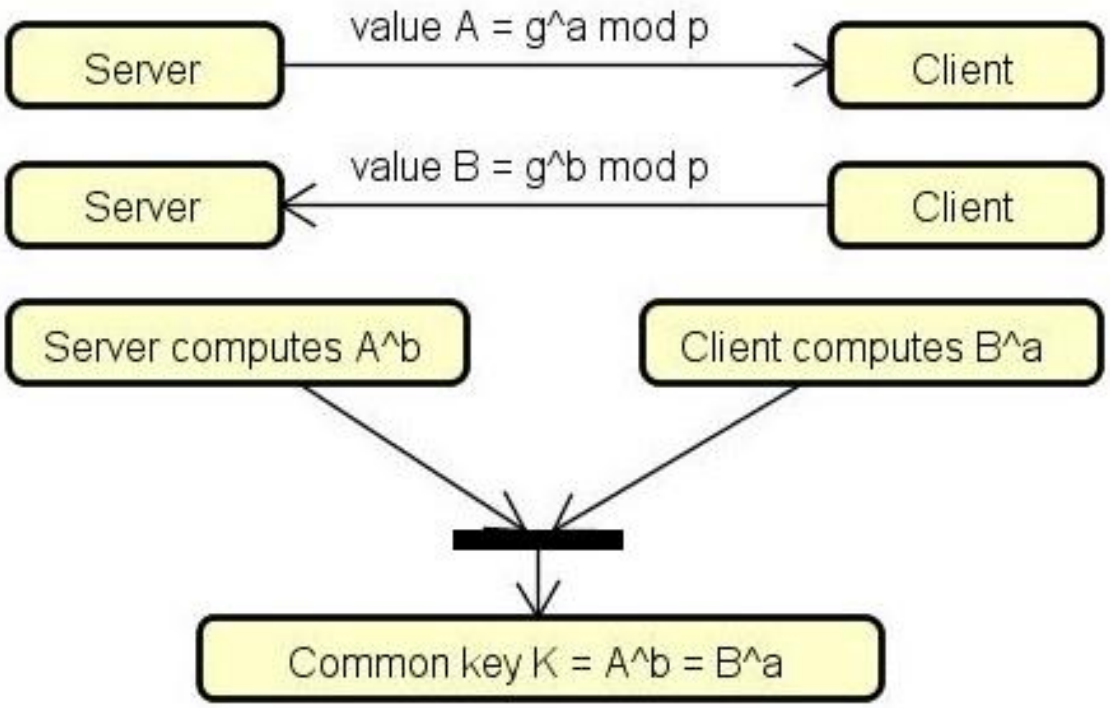
# SSS


Generate secret key
Share 1 - device
Share 2 - user
Share 3 - server
Restored secret key

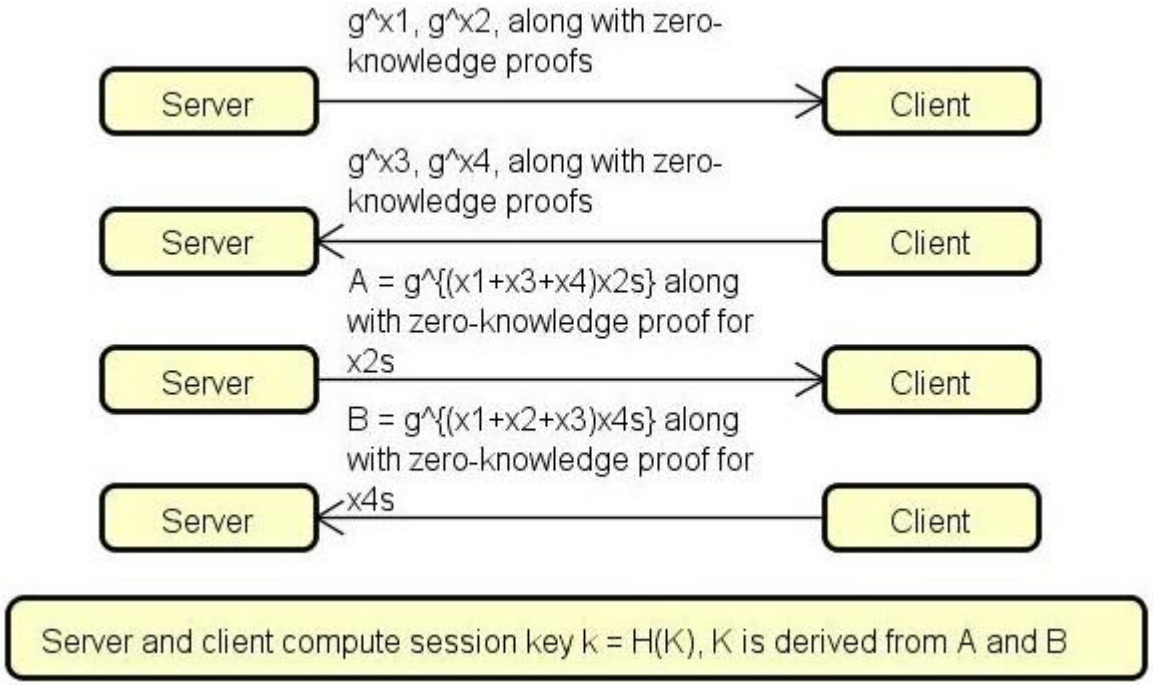
# AES-TOKEN


User login/password, PIN or timestamp value
Tokenization function (hash series)
One-time AES key
Protecting the transport of the distributed key

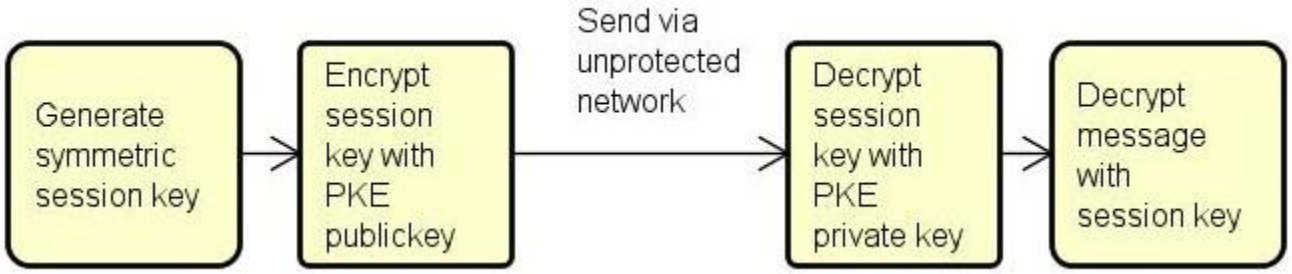
# Diffie-Hellman



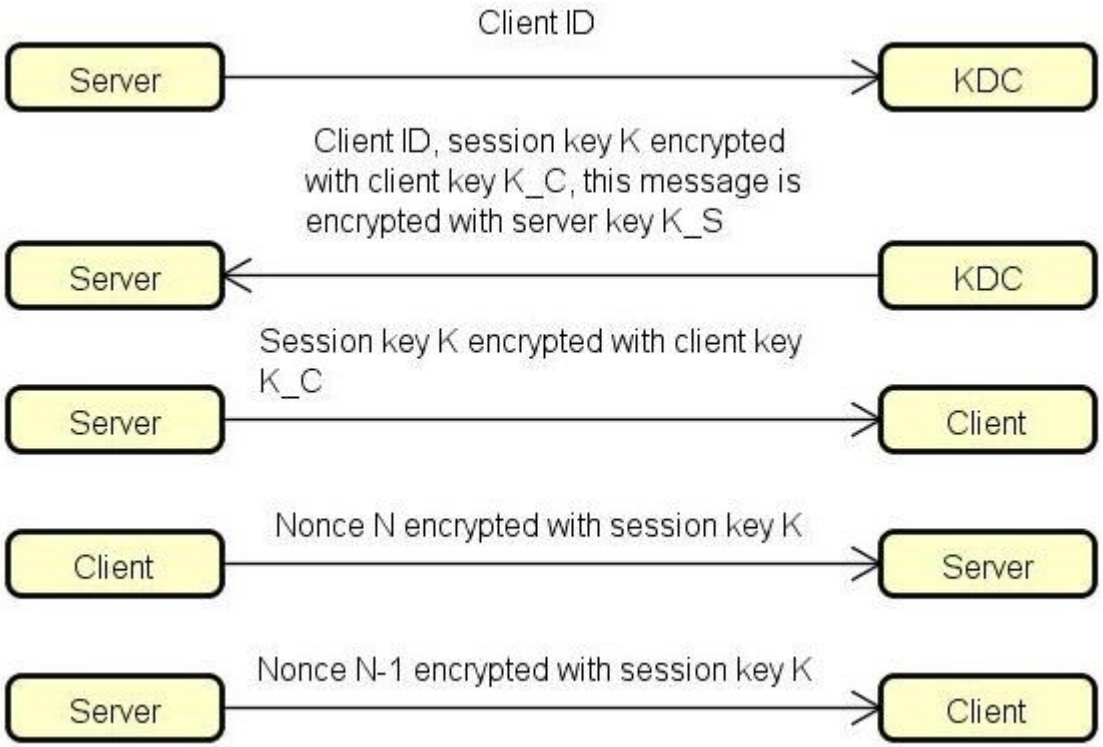
# PAKE


g^x1, g^x2, along with zero-knowledge proofs
Server
Client
g^x3, g^x4, along with zero-knowledge proofs
Server
Client
A = g^{(x1+x3+x4)x2s} along with zero-knowledge proof for x2s
Server
Client
B = g^{(x1+x2+x3)x4s} along with zero-knowledge proof for x4s
Server
Client
Server and client compute session key k = H(K), K is derived from A and B

# PKE


Generate symmetric session key
Encrypt session key with PKE publickey
Send via unprotected network
Decrypt session key with PKE private key
Decrypt message with session key

# KERBEROS


Client ID
Server
KDC
Client ID, session key K encrypted with client key K_C, this message is encrypted with server key K_S
Server
KDC
Session key K encrypted with client key K_C
Server
Client
Nonce N encrypted with session key K
Client
Server
Nonce N-1 encrypted with session key K
Server
Client