JAVASCRIPTCORE 实际应用

APATCH安全问题

- JSPatch 脚本执行权限很高，若被第三方篡改会带来很大安全问题。因此 APatch 和 HotFix 平台都对安全问题考虑良多。
- 从上图可看出，客户端从服务器下载 Patch 之前先要下载指定 Patch 配置信息即PatchInfo，其中包含了 Patch 文件密钥 file_token。 服务端： - 对 file_token 用 RSA 公钥加密。 - 对 PatchInfo 原始数据采用 HMacSha1 算法计算的哈希值，并将原始数据和哈希值serviceToken放在同一消息中传送给客户端。
- 客户端： - 使用 secret 计算所接收数据的哈希值。 - 检查计算所得的 HMAC 是否与传送的 HMAC 匹配。 - 只有PatchInfo通过校验匹配后才会去下载Patch。
- 另外，update patch 的接口已迁至 https，进一步保证了数据传输的安全。
- 本地存储的脚本被篡改的机会小很多，只在越狱机器上有点风险，对此 APatch SDK 对下载的脚本进行了AES对称加密，每次读取时： - 客户端使用 RSA 私钥解密 PatchInfo.file_token 获取 key 和 iv。 - 使用 key 和 iv 进行 AES 解密。
- 解密成功后的数据存储在 script 中，然后会调用 JSPatch 运行js脚本的接口：
- 至此，APatch 的工作已经完成，接下来具体的热修复工作就交给 JSPatch 了。

APATCH安全问题

- JSPatch 脚本执行权限很高，若被第三方篡改会带来很大安全问题。因此 APatch 和 HotFix 平台都对安全问题考虑良多。
- 从上图可看出，客户端从服务器下载 Patch 之前先要下载指定 Patch 配置信息即PatchInfo，其中包含了 Patch 文件密钥 file_token。 服务端： - 对 file_token 用 RSA 公钥加密。 - 对 PatchInfo 原始数据采用 HMacSha1 算法计算的哈希值，并将原始数据和哈希值serviceToken放在同一消息中传送给客户端。
- 客户端： - 使用 secret 计算所接收数据的哈希值。 - 检查计算所得的 HMAC 是否与传送的 HMAC 匹配。 - 只有PatchInfo通过校验匹配后才会去下载Patch。
- 另外，update patch 的接口已迁至 https，进一步保证了数据传输的安全。
- 本地存储的脚本被篡改的机会小很多，只在越狱机器上有点风险，对此 APatch SDK 对下载的脚本进行了AES对称加密，每次读取时： - 客户端使用 RSA 私钥解密 PatchInfo.file_token 获取 key 和 iv。 - 使用 key 和 iv 进行 AES 解密。
- 解密成功后的数据存储在 script 中，然后会调用 JSPatch 运行js脚本的接口：
- 至此，APatch 的工作已经完成，接下来具体的热修复工作就交给 JSPatch 了。

JAVASCRIPTCORE 头文件