

немного о структуре фингер принта

771,4867-4865-4866-52393-52392-49195-49199-49196-49200-49171-49172-156-157-47-53,0-23-65281-10-11-35-16-5-13-18-51-45-43-27-21,29-23-24,0-1-2

группы разбиваются запятыми

1 - номер tls; в поток пишется число

2 - список поддерживаемых сайферов; в поток пишутся просто числами, что дает возможность без проблем подменять сайферы не прописанные у cef

3 - список экстеншенов; каждый экстеншен представлен как структура со своим набором полей, соответственно чтобы использовать не прописанные у cef экстеншены, надо знать его структуру + плюс реализация их записи и чтения в поток

4 - набор значений для 10 экстеншена (присутствует если в экстеншенах есть 10)

5 - набор значений для 11 экстеншена (присутствует если в экстеншенах есть 11)

что из этого следует

сайферы мы можем писать любые, экстеншены только те что в коде у cef описаны, если нужен доп экстеншен придется искать где то его структуру и добавлять в cef.

..\chromium\src\third_party\boringssl\src\ssl\ssl_cipher.cc - содержит код с поддерживаемыми сайферами реализованными в cef

```
151 | #include <openssl/stack.h>
152 |
153 | #include "../crypto/internal.h"
154 | #include "internal.h"
155 | #include "net/ssl/ssl_config.h"
156 |
157 | namespace bssl {
158 |
159 | // kCiphers is an array of all supported ciphers, sorted by id.
160 | static constexpr SSL_CIPHER kCiphers[] = {
161 |     // The RSA ciphers
162 |     // Cipher 02
163 |     {
164 |         SSL3_TXT_RSA_NULL_SHA,
165 |         "TLS_RSA_WITH_NULL_SHA",
166 |         SSL3_CK_RSA_NULL_SHA,
167 |         SSL_kRSA,
168 |         SSL_aRSA,
169 |         SSL_eNULL,
170 |         SSL_SHA1,
171 |         SSL_HANDSHAKE_MAC_DEFAULT,
172 |     },
173 |
174 |     // Cipher 0A
175 |     {
```

..\chromium\src\third_party\boringssl\src\ssl\t1_lib.cc - содержит код с поддерживаемыми экстеншенами

```

2754     return true;
2755
2756
2757     // kExtensions contains all the supported extensions.
2758     static const struct tls_extension kExtensions[] = {
2759     {
2760         TLSEXT_TYPE_renegotiate,
2761         NULL,
2762         ext_ri_add_clienthello,
2763         ext_ri_parse_serverhello,
2764         ext_ri_parse_clienthello,
2765         ext_ri_add_serverhello,
2766     },
2767     {
2768         TLSEXT_TYPE_server_name,
2769         NULL,
2770         ext_sni_add_clienthello,
2771         ext_sni_parse_serverhello,
2772         ext_sni_parse_clienthello,
2773         ext_sni_add_serverhello,
2774     },
2775     {
2776         TLSEXT_TYPE_extended_master_secret,
2777         NULL,
2778         ext_ems_add_clienthello,
2779         ext_ems_parse_serverhello,
2780         ext_ems_parse_clienthello,
2781         ext_ems_add_serverhello,
2782     }

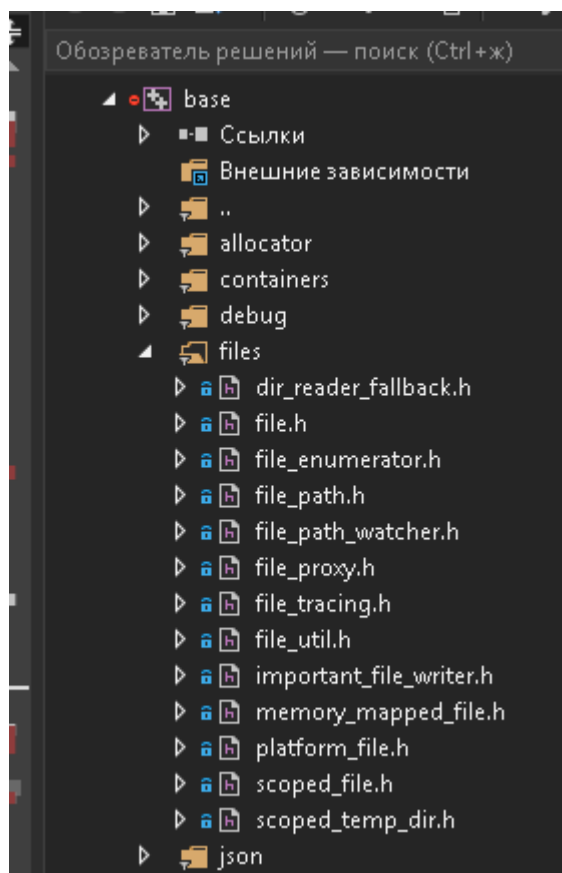
```

первый параметр собственно сам айдишник, дальше набор функций, которые как видно из названий реализуют функционал по инициализации, чтения, и парсинга экстеншена, если будет добавляться свой экстеншен то придется свои дописывать.

Работа с диском

убрал проверки из функций по работе с файловой системой.

весь функционал находится тут:



комментил вызов вот этой функции AssertBlockingAllowed();

```
792
793 FILE* OpenFile(const FilePath& filename, const char* mode) {
794     // 'N' is unconditionally added below, so be sure there is not one already
795     // present before a comma in |mode|.
796     DCHECK(
797         strchr(mode, 'N') == nullptr ||
798         (strchr(mode, ',') != nullptr && strchr(mode, 'N') > strchr(mode, ',')));
799     //AssertBlockingAllowed();
800     string16 w_mode = ASCIIToUTF16(mode);
801     AppendModeCharacter(L'N', &w_mode);
802     return _wfsopen(filename.value().c_str(), w_mode.c_str(), _SH_DENYNO);
803 }
804
```

Общая логика

для манипулирования фингер принтом создал класс SSL_ja3 (находиться
...\chromium\src\net\ssl\ssl_config.h\cc)

```

18
19 class SSL_ja3 {
20 public:
21     static SSL_ja3* GetInstance() {
22         return base::Singleton<SSL_ja3, base::LeakySingletonTraits<SSL_ja3>>::get();
23     }
24
25 public:
26     void InitFromFile();
27     void InitForTesting();
28     void InitForString(std::string str);
29     void LogMessage(std::string str);
30
31     uint16_t GetVersion() { return version_; }
32     void SetVersion(uint16_t version) { version_ = version; }
33
34 private:
35     friend struct base::DefaultSingletonTraits<SSL_ja3>;
36     SSL_ja3();
37     std::vector<std::string> split_string(std::string str, char delim);
38     std::vector<uint16_t> convert_str_to_unit16(std::vector<std::string>& str);
39     std::vector<uint8_t> convert_str_to_unit8(std::vector<std::string>& str);
40
41 public:
42     bool need_check;
43     uint16_t version_;
44     std::vector<uint16_t> cipher_suites_;
45     std::vector<uint16_t> custom_ext_;
46     std::vector<uint16_t> custom_supported_group_list_;
47     std::vector<uint8_t> custom_points_;
48
49 private:
50     bool was_init_;
51     base::File file_log_;
52 };
53

```

реализован как сингл инстанс.

функция InitFromFile() - зачитывает конфиг из файл

```

24
25 void SSL_ja3::InitFromFile() {
26     if (was_init_ == false) {
27         base::FilePath sett_file = base::FilePath(FILE_PATH_LITERAL("finger.txt"));
28         std::string finger_data;
29         if (base::ReadFileToString(sett_file, &finger_data) == true) {
30             // TODO add code, init params from code
31             InitForString(finger_data);
32             //InitForTesting();
33             was_init_ = true;
34         }
35     }
36 }
37
38 void SSL_ja3::InitForString(std::string str) {
39
40     std::vector<std::string> first = split_string(str, ',');
41
42     need_check = first[0] == "1";
43
44     version_ = std::stoi(first[1]);
45
46     std::vector<std::string> ciphers = split_string(first[2], '-');
47     cipher_suites_ = convert_str_to_unit16(ciphers);
48
49     std::vector<std::string> exts = split_string(first[3], '-');
50     custom_ext_ = convert_str_to_unit16(exts);
51
52     std::vector<std::string> groups_lists = split_string(first[4], '-');
53     custom_supported_group_list_ = convert_str_to_unit16(groups_lists);
54
55     std::vector<std::string> points_lists = split_string(first[5], '-');
56     custom_points_ = convert_str_to_unit8(points_lists);
57 }
58

```

логика достаточно тривиальная
смотрим читался ли конфиг, если нет то читает его из файла и парсит считанную строку по соответствующим полям класса с которыми в дальнейшем по коду работаем.

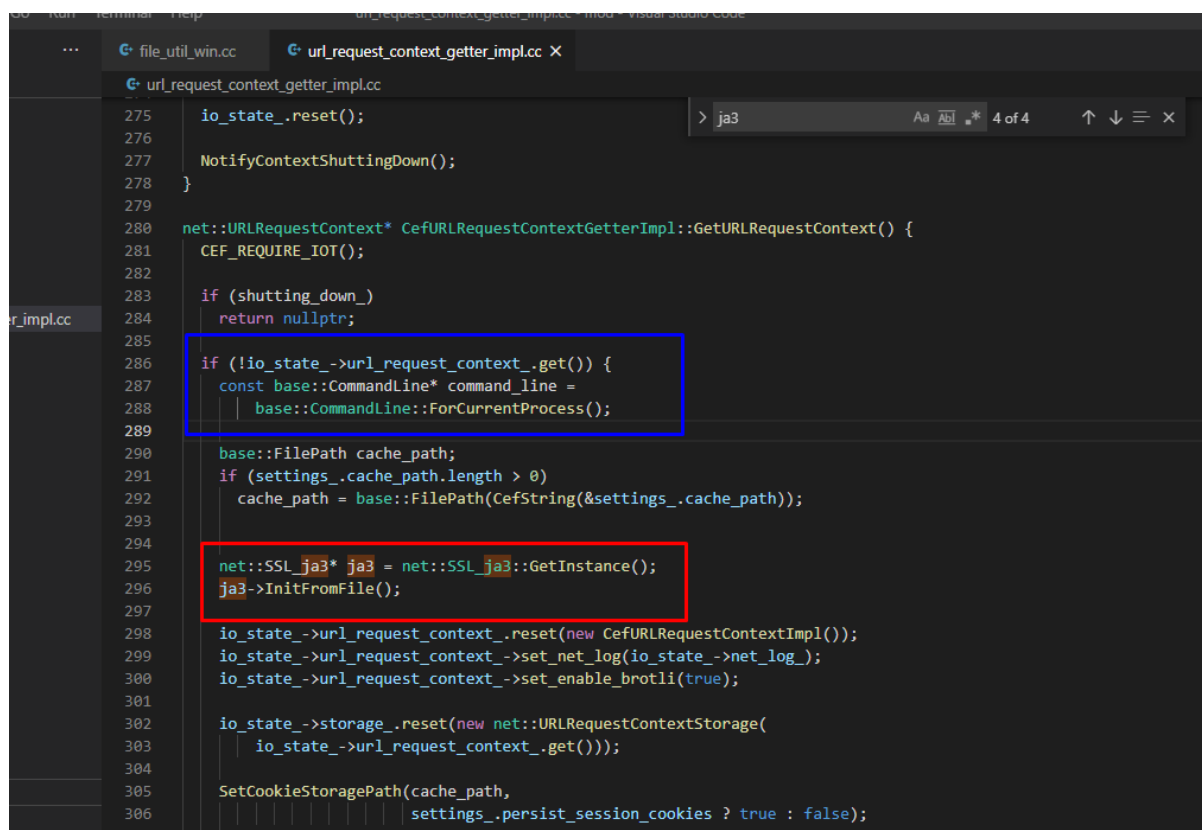
Структура где что менял:

url_request_context_getter_impl.cc

функция `net::URLRequestContext*`

`CefURLRequestContextGetterImpl::GetURLRequestContext()`

тут отправная точка где создается SSL_ja3 и зачитываются данные из конфига



```
275 io_state_.reset();
276
277 NotifyContextShuttingDown();
278 }
279
280 net::URLRequestContext* CefURLRequestContextGetterImpl::GetURLRequestContext() {
281   CEF_REQUIRE_IOT();
282
283   if (shutting_down_)
284     return nullptr;
285
286   if (!io_state_->url_request_context_.get()) {
287     const base::CommandLine* command_line =
288       base::CommandLine::ForCurrentProcess();
289
290     base::FilePath cache_path;
291     if (settings_.cache_path.length > 0)
292       cache_path = base::FilePath(CefString(&settings_.cache_path));
293
294     net::SSL_ja3* ja3 = net::SSL_ja3::GetInstance();
295     ja3->InitFromFile();
296
297     io_state->url_request_context_.reset(new CefURLRequestContextImpl());
298     io_state->url_request_context->set_net_log(io_state->net_log_);
299     io_state->url_request_context->set_enable_brotli(true);
300
301     io_state->storage_.reset(new net::URLRequestContextStorage(
302       io_state->url_request_context_.get()));
303
304     SetCookieStoragePath(cache_path,
305       settings_.persist_session_cookies ? true : false);
```

в этой же функции судя по коду доступна командная строка (синий прямоугольник на картинке выше), но это код выполняет в subprocess и поэтому эта командная строка отличается от оригинальной которая задается главному экзешнику, но в любом случае при желании можно протянуть свои параметры, просто надо расковырять код где эти subprocess стартуют.

что менял для манипуляции сайфер листами
файл ssl_cipher.cc

```
bool ssl_create_cipher_list(UniquePtr<SSLCipherPreferenceList>
*out_cipher_list,
                           const char *rule_str, bool strict)
```

эта функция по запросу формирует список доступных сайферов
из нее вызывается ssl_cipher_collect_ciphers - в которой формируется список и далее идет сортировка по алгоритму гугла, так вот сортировку я закоментировал

```

bool ssl_create_cipher_list(UniquePtr<SSLCipherPreferenceList> *out_cipher_list,
    const char *rule_str, bool strict) {
    // Return with error if nothing to do.
    if (rule_str == NULL || out_cipher_list == NULL) {
        return false;
    }

    // Now we have to collect the available ciphers from the compiled in ciphers.
    // We cannot get more than the number compiled in, so it is used for
    // allocation.
    Array<CIPHER_ORDER> co_list;
    CIPHER_ORDER *head = nullptr, *tail = nullptr;
    if (!ssl_cipher_collect_ciphers(&co_list, &head, &tail)) {
        return false;
    }

    //=====>
    //// Now arrange all ciphers by preference:
    //// TODO(davidben): Compute this order once and copy it.

    //// Everything else being equal, prefer ECDHE_ECDSA and ECDHE_RSA over other
    //// key exchange mechanisms
    // ssl_cipher_apply_rule(0, SSL_kECDHE, SSL_aECDSA, ~0u, ~0u, 0, CIPHER_ADD,
    // -1,
    // false, &head, &tail);
    // ssl_cipher_apply_rule(0, SSL_kECDHE, ~0u, ~0u, ~0u, 0, CIPHER_ADD, -1,
    // false,
    // &head, &tail);

```

закомментированный код находится между комментариями

//=====>

следующая функция

```

static bool ssl_cipher_collect_ciphers(Array<CIPHER_ORDER> *out_co_list,
    CIPHER_ORDER **out_head,
    CIPHER_ORDER **out_tail)

```

тут я добавил фильтрация сайфиров по умолчанию через заданные пользователем
то есть в итоговом наборе сайфиров останутся только те которые присутствуют из
коробки и заданном пользователем

```

714 //printf("hello from ssl_cipher_collect_ciphers");
715 //for (const SSL_CIPHER &foundCipher : kCiphers) {
716 // uint16_t id_cipher = ssl_cipher_get_value(&foundCipher);
717 //}
718
719 //uint16_t custom_cipher_list[] = {
720 // 4865, 4866, 4867, 49196, 49195, 49188, 49187, 49162, 49161,
721 // 52393, 49200, 49199, 49192, 49191, 49172, 49171, 52392, 157,
722 // 156, 61, 60, 53, 47, 49160, 49170, 10};
723 net::SSL_ja3 *ja3 = net::SSL_ja3::GetInstance();
724
725 size_t co_list_num = 0;
726
727 for (uint16_t cp : ja3->cipher_suites_) {
728     SSL_CIPHER *cipher = NULL;
729     for (const SSL_CIPHER &foundCipher : kCiphers) {
730         uint16_t id_cipher = ssl_cipher_get_value(&foundCipher);
731         if (cp == id_cipher) {
732             cipher = const_cast<SSL_CIPHER *>(&foundCipher);
733             break;
734         }
735     }
736
737     if (cipher == NULL) {
738         continue;
739     }
740
741     if (cipher->algorithm_mkey != SSL_kGENERIC) {

```

файл handshake_client.cc

функция

```

int ssl_write_client_hello(SSL_HANDSHAKE *hs) {
    SSL *const ssl = hs->ssl;

```

тут происходит формирования clienthello.


```

4
5 int ssl_write_client_hello(SSL_HANDSHAKE *hs) {
6     SSL *const ssl = hs->ssl;
7
8     net::SSL_ja3 *ja3 = net::SSL_ja3::GetInstance();
9     uint16_t v1 = ja3->GetVersion();
10
11     ScopedCBB cbb;
12     CBB body;
13     if (!ssl->method->init_message(ssl, cbb.get(), &body, SSL3_MT_CLIENT_HELLO)) {
14         return 0;
15     }
16
17     CBB child;
18     if (!CBB_add_u16(&body, ja3->version_ /*hs->client_version*/) ||
19         !CBB_add_bytes(&body, ssl->s3->client_random, SSL3_RANDOM_SIZE) ||
20         !CBB_add_u8_length_prefixed(&body, &child)) {
21         return 0;
22     }
23

```

подменили версию tls из конфига, далее нам интересен вызов двух функций

```

    }

    if (SSL_is_dtls(ssl)) {
        if (!CBB_add_u8_length_prefixed(&body, &child) ||
            !CBB_add_bytes(&child, ssl->d1->cookie, ssl->d1->cookie_len)) {
            return 0;
        }
    }

    size_t header_len =
        SSL_is_dtls(ssl) ? DTLS1_HM_HEADER_LENGTH : SSL3_HM_HEADER_LENGTH;
    if (!ssl_write_client_cipher_list(hs, &body) ||
        !CBB_add_u8(&body, 1 /* one compression method */) ||
        !CBB_add_u8(&body, 0 /* null compression */) ||
        !ssl_add_clienthello_tlsext(hs, &body, header_len + CBB_len(&body))) {
        return 0;
    }

    Array<uint8_t> msg;
    if (!ssl->method->finish_message(ssl, cbb.get(), &msg)) {
        return 0;
    }

    // Now that the length prefixes have been computed, fill in the placeholder
    // PSK binder.
    if (hs->needs_psk_binder &&
        !tls13_write_psk_binder(hs, msg.data(), msg.size())) {
        return 0;
    }
}

```

рассмотрим

```
static int ssl_write_client_cipher_list(SSL_HANDSHAKE *hs, CBB *out)
```

на данном этапе у нас есть на руках список сайферов отфильтрованные через наши пользовательские сайферы и без сортировки, собственно эта функция пишет этот список в поток при условии что у нас не выставлено в конфиге чтобы писать сайферы как есть

```
static int ssl_write_client_cipher_as_is(SSL_HANDSHAKE *hs, CBB *out) {
    CBB child;
    if (!CBB_add_u16_length_prefixed(out, &child)) {
        return 0;
    }

    net::SSL_ja3 *ja3 = net::SSL_ja3::GetInstance();
    for (uint16_t cipher : ja3->cipher_suites_) {
        if (!CBB_add_u16(&child, cipher)) {
            return 0;
        }
    }

    return CBB_flush(out);
}
```

```
static int ssl_write_client_cipher_list(SSL_HANDSHAKE *hs, CBB *out) {
```

```
    net::SSL_ja3 *ja3 = net::SSL_ja3::GetInstance();
    if (ja3->need_check == false) {
        return ssl_write_client_cipher_as_is(hs, out);
    }
}
```

```
SSL *const ssl = hs->ssl;
```

```
uint32_t mask_a, mask_k;
```

```
ssl_get_client_disabled(hs, &mask_a, &mask_k);
```

```
CBB child;
```

```
if (!CBB_add_u16_length_prefixed(out, &child)) {
    return 0;
}
```

```
// Add a fake cipher suite. See draft-davidben-tls-grease-01.
```

```
if (ssl->ctx->grease_enabled &&
    !CBB_add_u16(&child, ssl_get_grease_value(hs, ssl_grease_cipher))) {
    return 0;
}
```

```
// Add TLS 1.3 ciphers. Order ChaCha20-Poly1305 relative to AES-GCM based on
// hardware support.
```

```
if (hs->max_version >= TLS1_3_VERSION) {
    if (!EVP_has_aes_hardware() &&
        !CBB_add_u16(&child, TLS1_CK_CHACHA20_POLY1305_SHA256 & 0xffff)) {
        return 0;
    }
}
```

по сути у нас получилось два режима
первый пишет скажем так причесанные сайферы (фильтрованный список)
второй это просто записать сайферы какие указал пользователь
за второй режим отвечает написанная мною функция
static int ssl_write_client_cipher_as_is(SSL_HANDSHAKE *hs, CBB *out) - логика там
достаточно тривиальная

Далее рассмотрим вторую функцию

```
bool ssl_add_clienthello_tlsexth(SSL_HANDSHAKE *hs, CBB *out,  
                                size_t header_len)
```

в ней происходит запись экстеншенов в поток
реализация находится в файле t1_lib.cc

```
}  
  
bool ssl_add_clienthello_tlsexth(SSL_HANDSHAKE *hs, CBB *out,  
                                size_t header_len) {  
    //<Siv_code>  
    net::SSL_ja3 *ja3 = net::SSL_ja3::GetInstance();  
    uint16_t prev_max_version = hs->max_version;  
    hs->max_version = 772; // need for using pre last parameters  
    //</Siv_code>  
  
    SSL *const ssl = hs->ssl;  
    CBB extensions;  
    if (!CBB_add_u16_length_prefixed(out, &extensions)) {  
        OPENSSL_PUT_ERROR(SSL, ERR_R_INTERNAL_ERROR);  
        return false;  
    }  
  
    hs->extensions.sent = 0;
```

первым делом я искусственно занизил версию для tls - иначе часть старых
экстеншенов просто игнорировались кодом, записывал экстеншены и потом опять
восстановил прежнюю версию.

```
    // last_was_empty = (bytes_written == 4);  
    //}  
    //<Siv_code>  
    hs->max_version = prev_max_version;  
    //</Siv_code>  
  
    if (ssl->ctx->options_enabled) {
```

далее на картинке идет кусок где я фильтрую экстеншены указанные пользователем с
экстеншенами которые поддерживает хрониум

```

//<Siv_code>
// uint16_t custom_ext[] = {65281, 0, 23, 13, 5, 18, 16, 11, 51, 45, 43, 10,
// 21}; uint16_t custom_supported_group_list[] = {29, 23, 24, 25}; size_t sz =
// OPENSSL_ARRAY_SIZE(custom_supported_group_list);
size_t sz = ja3->custom_supported_group_list.size();
hs->config->supported_group_list.Init(sz);
for (size_t i = 0; i < sz; i++) {
    hs->config->supported_group_list[i] = ja3->custom_supported_group_list[i];
}
//<\Siv_code>
bool last_was_empty = false;
for (size_t f = 0; f < ja3->custom_ext.size(); f++) {
    for (size_t j = 0; j < kNumExtensions; j++) {
        if (kExtensions[j].value == ja3->custom_ext[f]) {
            const size_t len_before = CBB_len(&extensions);
            if (!kExtensions[j].add_clienthello(hs, &extensions)) {
                OPENSSL_PUT_ERROR(SSL, SSL_R_ERROR_ADDING_EXTENSION);
                ERR_add_error_dataf("extension %u", (unsigned)kExtensions[j].value);
                return false;
            }

            const size_t bytes_written = CBB_len(&extensions) - len_before;
            if (bytes_written != 0) {
                hs->extensions.sent |= (1u << j);
            }
            // If the difference in lengths is only four bytes then the extension
            // had an empty body.
            last_was_empty = (bytes_written == 4);
            break;
        }
    }
}
}

```

то есть алгоритм следующий получился

понижил версию tls

отфильтровал пользовательский список экстеншенов и записал только те что присутствуют и в хранилище и у пользователя.

вернул версию tls

еще стоит обратить внимание на следующий код

```

//<Siv_code>
// uint16_t custom_ext[] = {65281, 0, 23, 13, 5, 18, 16, 11, 51, 45, 43, 10,
// 21}; uint16_t custom_supported_group_list[] = {29, 23, 24, 25}; size_t sz =
// OPENSSL_ARRAY_SIZE(custom_supported_group_list);
size_t sz = ja3->custom_supported_group_list.size();
hs->config->supported_group_list.Init(sz);
for (size_t i = 0; i < sz; i++) {
    hs->config->supported_group_list[i] = ja3->custom_supported_group_list[i];
}
//</Siv_code>

bool last_was_empty = false;
for (size_t f = 0; f < ja3->custom_ext.size(); f++) {
    for (size_t j = 0; j < kNumExtensions; j++) {
        if (kExtensions[j].value == ja3->custom_ext[f]) {
            const size_t len_before = CBB_len(&extensions);
            if (!kExtensions[j].add_clienthello(hs, &extensions)) {
                OPENSSL_PUT_ERROR(SSL, SSL_R_ERROR_ADDING_EXTENSION);
                ERR_add_error_dataf("extension %u", (unsigned)kExtensions[j].value);
                return false;
            }

            const size_t bytes_written = CBB_len(&extensions) - len_before;
            if (bytes_written != 0) {
                hs->extensions.sent |= (1u << j);
            }
            // If the difference in lengths is only four bytes then the extension
            // had an empty body.
            last_was_empty = (bytes_written == 4);
            break;
        }
    }
}
}

```

этот код необходим чтобы указать состав для 10 экстеншена

```

1747 //
1748 // https://tools.ietf.org/html/rfc4492#section-5.1.2
1749
1750 static bool ext_ec_point_add_extension(SSL_HANDSHAKE *hs, CBB *out) {
1751     CBB contents, formats;
1752     if (!CBB_add_u16(out, TLSEXT_TYPE_ec_point_formats) ||
1753         !CBB_add_u16_length_prefixed(out, &contents) ||
1754         !CBB_add_u8_length_prefixed(&contents, &formats)){
1755         return false;
1756     }
1757 }
1758
1759 net::SSL_ja3 *ja3 = net::SSL_ja3::GetInstance();
1760 for (uint8_t group : ja3->custom_points_) {
1761     if (!CBB_add_u8(&formats, group)) {
1762         return false;
1763     }
1764 }
1765
1766 return CBB_flush(out);
1767

```

этим кодом я формирую состав для 11 экстеншена

Итого по формированию ClientHello вроде все

Следующий что рассмотрим это обработка ответа от сервера
файл t1_lib.cc

функция

```
static bool ssl_scan_serverhello_tlsexth(SSL_HANDSHAKE *hs, CBS *cbs,  
int *out_alert) {
```

тут зачитывают экстеншены выбранные сервером для работы

тут я закоментировал проверку на предмет соответствия что послалось и что мы
приняли обратно

```
3288  
3289 // Decode the next extension.  
3290 if (!CBS_get_u16(&extensions, &type) ||  
3291     !CBS_get_u16_length_prefixed(&extensions, &extension)) {  
3292     *out_alert = SSL_AD_DECODE_ERROR;  
3293     return false;  
3294 }  
3295  
3296 unsigned ext_index;  
3297 const struct tls_extension *const ext =  
3298     | tls_extension_find(&ext_index, type);  
3299  
3300 if (ext == NULL) {  
3301     OPENSSL_PUT_ERROR(SSL, SSL_R_UNEXPECTED_EXTENSION);  
3302     ERR_add_error_dataf("extension %u", (unsigned)type);  
3303     *out_alert = SSL_AD_UNSUPPORTED_EXTENSION;  
3304     return false;  
3305 }  
3306  
3307 static_assert(kNumExtensions <= sizeof(hs->extensions.sent) * 8,  
3308             "too many bits");  
3309  
3310 //if (!(hs->extensions.sent & (1u << ext_index))) {  
3311 //    // If the extension was never sent then it is illegal.  
3312 //    OPENSSL_PUT_ERROR(SSL, SSL_R_UNEXPECTED_EXTENSION);  
3313 //    ERR_add_error_dataf("extension :%u", (unsigned)type);  
3314 //    *out_alert = SSL_AD_UNSUPPORTED_EXTENSION;  
3315 //    return false;  
3316 //}  
3317  
3318 received |= (1u << ext_index);  
3319  
3320 uint8_t alert = SSL_AD_DECODE_ERROR;  
3321 if (!ext->parse_serverhello(hs, &alert, &extension)) {  
3322     OPENSSL_PUT_ERROR(SSL, SSL_R_ERROR_PARSING_EXTENSION);  
3323     ERR_add_error_dataf("extension %u", (unsigned)type);  
3324     *out_alert = alert;  
3325     return false;  
3326 }  
3327 }  
3328  
3329 for (size_t i = 0; i < kNumExtensions; i++) {
```

то есть по сути хром из коробки всегда шлет один и тот же набор экстеншенов
и пытается этот момент контролировать, но так как мы меняем и состав и порядок
экстеншенов то эту проверку пришлось убрать иначе постоянно ругался на не
соответствие что хотел и что получили

второй момент это бработка полученных сайферов от сервера тут я просто добавил вывод в лог если в обратку сервер выберет какой нить сайфер который не поддерживается текущей реализацией браузер, а получиться это может в тот момент когда мы шлем серверну сайферы как они есть в конфиге и как вариант сервер может выбрать сайфер который не описан в хrome.

файл

функция

static enum ssl_hs_wait_t do_read_server_hello(SSL_HANDSHAKE *hs)

```
654     SSL_get_session(ssl, NULL);
655     if (!ssl_get_new_session(hs, 0 /* client */)) {
656         ssl_send_alert(ssl, SSL3_AL_FATAL, SSL_AD_INTERNAL_ERROR);
657         return ssl_hs_error;
658     }
659     // Note: session_id could be empty.
660     hs->new_session->session_id_length = CBS_len(&session_id);
661     OPENSSL_memcpy(hs->new_session->session_id, CBS_data(&session_id),
662                   CBS_len(&session_id));
663 }
664
665 const SSL_CIPHER *cipher = SSL_get_cipher_by_value(cipher_suite);
666 if (cipher == NULL) {
667     // unknown cipher
668     net::SSL_ja3 *ja3 = net::SSL_ja3::GetInstance();
669     ja3->LogMessage("unknown cipher: " + std::to_string(cipher_suite));
670
671     OPENSSL_PUT_ERROR(SSL, SSL_R_UNKNOWN_CIPHER_RETURNED);
672     ssl_send_alert(ssl, SSL3_AL_FATAL, SSL_AD_ILLEGAL_PARAMETER);
673     return ssl_hs_error;
674 }
675
676 // The cipher must be allowed in the selected version and enabled.
677 uint32_t mask_a, mask_k;
678 ssl_get_client_disabled(hs, &mask_a, &mask_k);
679 if ((cipher->algorithm_mkey & mask_k) || (cipher->algorithm_auth & mask_a) ||
680     SSL_CIPHER_get_min_version(cipher) > ssl_protocol_version(ssl) ||
681     SSL_CIPHER_get_max_version(cipher) < ssl_protocol_version(ssl) ||
682     !sk_SSL_CIPHER_find(SSL_get_ciphers(ssl), NULL, cipher)) {
683     OPENSSL_PUT_ERROR(SSL, SSL_R_WRONG_CIPHER_RETURNED);
684     ssl_send_alert(ssl, SSL3_AL_FATAL, SSL_AD_ILLEGAL_PARAMETER);
685     return ssl_hs_error;
686 }
```

то есть если мы словили такую ситуацию то мы пишем об этом информацию в лог и выводим в браузере ошибку.

В целом вроде логику описал