Dokumentacja Techniczna

Opis modyfikacji firmware czytnika UKM-900 na potrzeby parkingu P&R MKA

1. Wprowadzenie

Dokument zawiera opis modyfikacji firmware czytnika UKM-900 wykonanych na potrzeby odczytu i weryfikacji kart MKA oraz weryfikacji kodów 2D wykorzystywanych do autoryzacji wjazdu na parking P&R.
Karta MKA jest kartą JCOP posiadającą dedykowany aplet MKA oraz emulacje kart Mifare Classic. W celu wykorzystania wszystkich zalet karty, zapewnienia najwyższego poziomu bezpieczeństwa oraz uniemożliwienia kopiowania biletów pomiędzy nośnikami niezbędna jest autoryzacja do apletu MKA.

2. Wykorzystanie modułu SAM AV2.

Czytnik został dostosowany do obsługi modułów SAM AV2 wykorzystywanych w systemie MKA. Zmiany polegały na:

- wprowadzeniu dedykowanego klucza umożliwiającego odblokowanie i użycie modułu SAM AV2
- zmianie numeru i wersji klucza do którego odwołuje się czytnik podczas procesu odblokowywania modułu AV2

3. Weryfikacja karty MKA

Na potrzeby weryfikacji oryginalności karty z systemu MKA została utworzona dedykowana funkcjonalność, który umożliwia autoryzację do apletu MKA. Autoryzacja jest wykonywana przy wykorzystaniu kluczy i mechanizmów znajdujących się w module AV2.

Proces autoryzacji czytnika do apletu można podzielić na następujące etapy:

- Aktywacja apletu na karcie zgodnie z komendami protokołu ISO-7816.
- Przesłanie komendę do apletu rozpoczynającą proces autoryzacji.
- Przesyła komendę do modułu SAM rozpoczynającą proces autoryzacji przy pomocy wskazanego numeru i wersji klucza.
- Czytnik przekazuje wyzwania i odpowiedzi generowane w procesie autoryzacji pomiędzy apletem i modułem SAM.
- Czytnik odczytuje z modułu SAM informację o pozytywnym lub negatywnym wyniku autoryzacji z apletem.

Czytnik na podstawie wyniku autoryzacji oraz zapisanych danych na karcie decyduje czy bilety na karcie są oryginalne czy sfałszowane.

Tabela 1 przedstawia treść funkcji realizującej powyższą funkcjonalność.

```
mpk_res_t MKA_Card_AuthKey(cm_t *pCM, sn_t* sn, uint8_t jcop_key_idx, uint8_t sam_key_idx, uint8_t
sam_key_ver)
{
        cm_status_t cm_status;
        mpk_res_t status;
        phStatus_t ph_stat;

        uint8_t command[48]; uint8_t command_len;
        uint8_t response[64]; uint8_t response_len;
        uint8_t div_seq[MKA_JCOP_DIV_SEQ_LEN];
        if(jcop_key_idx > MAX_KEY_ENTRY_JCOP)
                return MPK_CR_WRONG_PARAM;
        if(sam_key_idx > MAX_KEY_ENTRY_SAM)
                return MPK_CR_WRONG_PARAM;
        memcpy(command, MKA_CMD_AUTH_GET_CHALLENGE_MASTER, sizeof(MKA_CMD_AUTH_GET_CHALLENGE_MASTER));
        command[ISO7816_P2_POS] = jcop_key_idx;
        response_len = 0;
        cm_status = CardManage_JCOP_Exchange(pCM, command, sizeof(MKA_CMD_AUTH_GET_CHALLENGE_MASTER),
response, &response_len);
        if ( (cm_status != CM_STAT_OK) || (response_len != MKA_JCOP_GET_CHALLENGE_LEN))
        {
                return MPK_CR_NO_FUNC;
        }
        memcpy(command, response, response_len);
        command_len = response_len;
        response_len = 0;
        MKA_Card_Auth_Prepare_DivSeq(div_seq, sizeof(div_seq), sn);
        ph_stat = phhalHw_SamAV2_Cmd_SAM_AuthenticatePICC_Part1(&phhalHw_SamAV2_DataParams,
                0x11,
                sam_key_idx,
                sam_key_ver,
                command,
                command_len,
                div_seq,
                MKA_JCOP_DIV_SEQ_LEN,
                response,
                &response_len);
        if( ph_stat != ( PH_COMP_HAL | PH_ERR_SUCCESS_CHAINING) )
        {
                return MPK_CR_SAM_ERR;
        }
        if(response_len != MKA_SAM_GET_CHALLENGE_LEN)
        {
                return MPK_CR_SAM_ERR;
        }
        memcpy(command, MKA_CMD_AUTH_EXTERNAL, sizeof(MKA_CMD_AUTH_EXTERNAL));
        memcpy(command + sizeof(MKA_CMD_AUTH_EXTERNAL), response, response_len);
        command_len = sizeof(MKA_CMD_AUTH_EXTERNAL) + response_len;
        response_len = 0;
        cm_status = CardManage_JCOP_Exchange(pCM, command, command_len, response, &response_len);

        if( (cm_status != CM_STAT_OK) || (response_len != MKA_JCOP_EXT_AUTH_LEN))
                return MPK_CR_NO_FUNC;

        ph_stat = phhalHw_SamAV2_Cmd_SAM_AuthenticatePICC_Part2(&phhalHw_SamAV2_DataParams,
                response,
                response_len);
        switch(ph_stat){
        case 0:
                status = MPK_CR_OK; break;
        case ( PH_COMP_HAL | PHHAL_HW_SAMAV2_ERR_CRYPTO):
                status = MPK_CR_WRONG_KEY; break;
        default:
                status = MPK_CR_SAM_ERR; break;
        }
        return status;
}
```

Tabela 1: Treść funkcji autoryzującej się do karty MKA

4. Weryfikacja kodów 2D

Na potrzeby weryfikacji oryginalności i poprawności kodów 2D odczytanych z aplikacji iMKA została utworzona dedykowana funkcjonalność, która umożliwia analizę kodu 2D. Weryfikacja jest wykonywana przy wykorzystaniu kluczy i mechanizmów znajdujących się w module SAM AV2.

Proces weryfikacji kodu 2D składa się z poszczególnych etapów:

- Przesłanie odczytanego kodu w dedykowanej komendzie do czytnika.
- Analizę składniową ciągu w celu wykluczenia kodów w których:
  - występują niedozwolone znaki
  - występują niedozwolone kombinacje znaków
  - brak jest wymaganych znaków
- Treść kodu jest wysyłana do modułu SAM, który na podstawie klucza niejawnego przechowywanego w module SAM oraz dedykowanego mechanizmu generuje klucz sesyjny
- Na podstawie klucza sesyjnego wyznaczane są sumy kontrolne z kodu 2D
- Wyznaczone sumy kontrolne są porównywane z sumami zapisanymi w kodzie 2D
- Czytnik zwraca odpowiedź z informacją o poprawności kodu 2D

Tabela 2 przedstawia treść funkcji realizującej powyższą funkcjonalność.

```
mpk_res_t   mka_verify_code2d(params_t   *DataParams,   uint8_t   *messages,   uint16_t
messages_len)
{
	uint8_t status = 0;
	struct code2d_t hmac_code;
	status = code2d_init(&hmac_code, (char*)messages, messages_len);
	if(status)
	{
		MakeLog_print_error(status);
		return status;
	}
	status = code2d_get_key(&hmac_code, DataParams);
	if(STAT_ERR_KEY_ENC_SAM == status)
	{
		DataParams->sam_error_flag = 1;
	}
	else
	{
		DataParams->sam_error_flag = 0;
	}
	if(status)
	{
		MakeLog_print_error(status);
		return status;
	}
	status = code2d_get_hmac_pos(&hmac_code);
	if(status)
	{
		MakeLog_print_error(status);
		return status;
	}
	status = code2d_calc_hmac(&hmac_code);
	if(status)
	{
		MakeLog_print_error(status);
		return status;
	}
	status = code2d_verify_hmac(&hmac_code);
	if(status)
	{
		MakeLog_print_error(status);
		return status;
	}
	MakeLog_print_error(STAT_OK);
	return 0;
}
```

Tabela 2: Treść funkcji analizującej poprawność kodu 2D